

Decidable Reasoning About Time in Finite-Domain Situation Calculus Theories

TIME'26

Till Hofmann, Stefan Schupp, Gerhard Lakemeyer

September 3, 2026



Outline

1. The Situation Calculus

1.1 Basic Action Theories

1.2 GOLOG

1.3 Applications

1.4 Decidability

2. Time in the Situation Calculus

2.1 Desiderata

2.2 Undecidability of Existing Approaches

3. Decidable Reasoning about Time in the Situation Calculus

3.1 Clocked Basic Action Theories

3.2 Regionalization

3.3 Time-Abstract Transition Systems

4. Realizability

5. Conclusion

The Situation Calculus¹

- ▶ First-order axiomatization of *actions and change*
- ▶ A *situation* is described as a *sequence of actions*, e.g.:

$$ePour(Mug, sPour(Mug), S_0)$$

is the situation after starting and ending to pour coffee into *Mug*

- ▶ *Fluents* have *situation arguments* and describe the *current state of the world*:

$$\neg IsFull(Mug, S_0)$$

$$IsFull(Mug, ePour(Mug, sPour(Mug), S_0))$$

¹John McCarthy. *Situations, Actions, and Causal Laws*. Technical Report. Stanford University, 1963; John McCarthy and Patrick J. Hayes. "Some Philosophical Problems from the Standpoint of Artificial Intelligence". In: *Mach. Intell.* 4 (1969); Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

The Coffee Robot

A *basic action theory* (BAT) axiomatizes the initial situation, and action preconditions and effects:

- ▶ A set $\mathcal{D}_0$ describes the initial situation:

$$\neg Brew(p, S_0), \neg Pour(p, m, S_0), \neg IsFull(o, S_0)$$

- ▶ Precondition axioms describe when actions are applicable:

$$Poss(sBrew(p), s) \equiv \neg Brew(p, s) \wedge \neg Pour(p, m, s)$$

$$Poss(eBrew(p), s) \equiv Brew(p, s)$$

$$Poss(sPour(p, m), s) \equiv IsFull(p, s) \wedge \neg Pour(p', m', s) \wedge \neg Brew(p, s)$$

$$Poss(ePour(p, m), s) \equiv Pour(p, m, s)$$

- ▶ Successor state axioms (SSAs) describe how actions *change* fluents:

$$IsFull(o, do(a, s)) \equiv a = eBrew(o) \vee \exists p. a = ePour(p, o) \vee IsFull(o, s)$$

$$Brew(p, do(a, s)) \equiv a = sBrew(p) \vee Brew(p, s) \wedge a \neq eBrew(p)$$

$$Pour(p, m, do(a, s)) \equiv a = sPour(p, m) \vee Pour(p, m, s) \wedge a \neq ePour(p, m)$$

High-Level Programming with GoLOG

- ▶ GoLOG: high-level programming language based on the situation calculus
- ▶ Imperative language tailored to specify agent behavior
- ▶ Atomic instructions: primitive actions, described in the situation calculus
- ▶ Program constructs:
 - $\phi?$: Test if ϕ is true
 - $\pi(x)$: Pick a value for variable x
 - δ^* : Repeat sub-program δ
 - $\delta_1|\delta_2$: Choose between δ_1 and δ_2
 - $\delta_1||\delta_2$: Interleaved concurrency
- ▶ Loops and conditionals can be defined as macros, e.g.,:

while ϕ do δ done $:= (\phi?; \delta)^* ; \neg\phi?$

```
sBrew(Pot); eBrew(Pot);  
while ( $\exists m \neg IsFull(m)$ ) do  
     $\pi m. sPour(Pot, m); ePour(Pot, m);$   
done
```

Applications of the Situation Calculus

Many problems can be formalized in the situation calculus:

► **Planning:**² given a BAT $\mathcal{D}$, does a sequence of actions satisfying a goal formula ϕ exist?

→ Planning is a *reachability problem*:

$$\mathcal{D} \models \exists s. \text{Exec}(s) \wedge \phi[s]$$

$\text{Exec}(s) \hat{=} s$ is a sequence of executable actions

²Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001; Jens Claßen. "Planning and Verification in the Agent Language Golog". PhD thesis. Aachen, Germany: RWTH Aachen University, 2013.

³Benjamin Zarriß and Jens Claßen. "Decidable Verification of Golog Programs over Non-Local Effect Actions". In: *AAAI*. 2016. doi: 10.1609/aaai.v30i1.10109; Jens Claßen. "Symbolic Verification of Golog Programs with First-Order BDDs". In: *KR*. 2018.

⁴Till Hofmann and Jens Claßen. "LTLf Synthesis on First-Order Agent Programs in Nondeterministic Environments". In: *AAAI*. 2025. doi: 10.1609/aaai.v39i14.33642; Giuseppe De Giacomo et al. "Reactive Synthesis for Golog Specifications in the Propositional Situation Calculus". In: *KR*. 2026.

Applications of the Situation Calculus

Many problems can be formalized in the situation calculus:

- **Planning:**² given a BAT $\mathcal{D}$, does a sequence of actions satisfying a goal formula ϕ exist?

→ Planning is a *reachability problem*:

$$\mathcal{D} \models \exists s. \text{Exec}(s) \wedge \phi[s]$$

$\text{Exec}(s) \hat{=} s$ is a sequence of executable actions

- **Verification:**³ given a BAT $\mathcal{D}$, a Golog program δ and a temporal formula Ψ (e.g., CTL*), does every execution of δ satisfy Ψ ?

$$\mathcal{D} \models \llbracket \delta \rrbracket \Psi$$

²Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001; Jens Claßen. "Planning and Verification in the Agent Language Golog". PhD thesis. Aachen, Germany: RWTH Aachen University, 2013.

³Benjamin Zarrieß and Jens Claßen. "Decidable Verification of Golog Programs over Non-Local Effect Actions". In: *AAAI*. 2016. doi: 10.1609/aaai.v30i1.10109; Jens Claßen. "Symbolic Verification of Golog Programs with First-Order BDDs". In: *KR*. 2018.

⁴Till Hofmann and Jens Claßen. "LTLf Synthesis on First-Order Agent Programs in Nondeterministic Environments". In: *AAAI*. 2025. doi: 10.1609/aaai.v39i14.33642; Giuseppe De Giacomo et al. "Reactive Synthesis for Golog Specifications in the Propositional Situation Calculus". In: *KR*. 2026.

Applications of the Situation Calculus

Many problems can be formalized in the situation calculus:

- **Planning:**² given a BAT $\mathcal{D}$, does a sequence of actions satisfying a goal formula ϕ exist?

→ Planning is a *reachability problem*:

$$\mathcal{D} \models \exists s. \text{Exec}(s) \wedge \phi[s]$$

Exec(s) $\hat{=}$ s is a sequence of executable actions

- **Verification:**³ given a BAT $\mathcal{D}$, a Golog program δ and a temporal formula Ψ (e.g., CTL*), does every execution of δ satisfy Ψ ?

$$\mathcal{D} \models \llbracket \delta \rrbracket \Psi$$

- **Synthesis:**⁴ given a BAT $\mathcal{D}$, a Golog program δ , and an LTLf goal Ψ , find a policy π that *resolves the nondeterminisms in δ* such that Ψ is always satisfied:

$$\mathcal{D} \models \llbracket \pi \rrbracket \Psi$$

²Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001; Jens Claßen. "Planning and Verification in the Agent Language Golog". PhD thesis. Aachen, Germany: RWTH Aachen University, 2013.

³Benjamin Zarrieß and Jens Claßen. "Decidable Verification of Golog Programs over Non-Local Effect Actions". In: *AAAI*. 2016. doi: 10.1609/aaai.v30i1.10109; Jens Claßen. "Symbolic Verification of Golog Programs with First-Order BDDs". In: *KR*. 2018.

⁴Till Hofmann and Jens Claßen. "LTLf Synthesis on First-Order Agent Programs in Nondeterministic Environments". In: *AAAI*. 2025. doi: 10.1609/aaai.v39i14.33642; Giuseppe De Giacomo et al. "Reactive Synthesis for Golog Specifications in the Propositional Situation Calculus". In: *KR*. 2026.

Decidable Fragments of the Situation Calculus

- ▶ As first-order logic is undecidable, many problems in the situation calculus are also undecidable
- ▶ Decidable fragments exist:
 - Restrict base logic to C^2 and syntactically constrain SSAs⁵
 - *Bounded situation calculus*: The number of “active” objects is bounded by a constant⁶
 - *Propositional situation calculus*: The set of objects O is fixed and finite⁷

⁵Benjamin Zarrieß and Jens Claßen. “Decidable Verification of Golog Programs over Non-Local Effect Actions”. In: *AAAI*. 2016. doi: 10.1609/aaai.v30i1.10109.

⁶Giuseppe De Giacomo et al. “Bounded Situation Calculus Action Theories”. In: *AIJ* 237 (Aug. 1, 2016). doi: 10.1016/j.artint.2016.04.006.

⁷Giuseppe De Giacomo et al. “Reactive Synthesis for Golog Specifications in the Propositional Situation Calculus”. In: *KR*. 2026.

Decidable Fragments of the Situation Calculus

- ▶ As first-order logic is undecidable, many problems in the situation calculus are also undecidable
- ▶ Decidable fragments exist:
 - Restrict base logic to C^2 and syntactically constrain SSAs⁵
 - *Bounded situation calculus*: The number of “active” objects is bounded by a constant⁶
 - *Propositional situation calculus*: The set of objects O is fixed and finite⁷
- ▶ In the following, we use the **propositional situation calculus**

⁵ Benjamin Zarriß and Jens Claßen. “Decidable Verification of Golog Programs over Non-Local Effect Actions”. In: *AAAI*. 2016. doi: 10.1609/aaai.v30i1.10109.

⁶ Giuseppe De Giacomo et al. “Bounded Situation Calculus Action Theories”. In: *AIJ* 237 (Aug. 1, 2016). doi: 10.1016/j.artint.2016.04.006.

⁷ Giuseppe De Giacomo et al. “Reactive Synthesis for Golog Specifications in the Propositional Situation Calculus”. In: *KR*. 2026.

Time in The Situation Calculus

Why add time to the situation calculus?

- ▶ Many domains require metric constraints that cannot be expressed with actions or discrete fluents:
 - A robot must arrive at a machine within a processing window
 - A robot gripper must stay closed for at least half a second before lifting

Time in The Situation Calculus

Why add time to the situation calculus?

- ▶ Many domains require metric constraints that cannot be expressed with actions or discrete fluents:
 - A robot must arrive at a machine within a processing window
 - A robot gripper must stay closed for at least half a second before lifting
- ▶ Why dense time?
 - Discretization forces a fixed granularity
 - Durations of physical processes do not adhere to the discretization
 - A coarse granularity loses precision
 - A fine granularity leads to an unnecessary blowup

Time in The Situation Calculus

Why add time to the situation calculus?

- ▶ Many domains require metric constraints that cannot be expressed with actions or discrete fluents:
 - A robot must arrive at a machine within a processing window
 - A robot gripper must stay closed for at least half a second before lifting
- ▶ Why dense time?
 - Discretization forces a fixed granularity
 - Durations of physical processes do not adhere to the discretization
 - A coarse granularity loses precision
 - A fine granularity leads to an unnecessary blowup
- ▶ Extending the running example:
 - If the robot brews coffee for more than 2 sec, the coffee becomes *strong*
 - Some users prefer strong coffee, while others prefer weak coffee

Existing Approaches: Real-Valued Fluents

Idea 1: add a real-valued sort *time* to the logic⁸, including arithmetic operations (+, −, ·, ÷), obtaining *real-valued BATs*

- In the coffee domain, we can add a fluent *brewStart(p)*:

$$\begin{aligned} \text{brewStart}(p, \text{do}(a, s)) = t &\equiv a = \text{sBrew}(p, t) \vee a \neq \text{sBrew}(p, t') \wedge \text{brewStart}(p, s) = t \\ \text{Strong}(p, \text{do}(a, s)) &\equiv \exists t. a = \text{eBrew}(p, t) \wedge t - \text{brewStart}(p, s) \geq 2 \\ &\quad \vee a \neq \text{eBrew}(p, t') \wedge \text{Strong}(p, s) \end{aligned}$$

Existing Approaches: Real-Valued Fluents

- ▶ We can do arbitrary arithmetic operations on the reals

⇒ Unsurprisingly:

Theorem

The reachability problem for a real-valued BAT is undecidable.

- ▶ Proof idea: Given a 2-counter machine (2CM) $\mathcal{M}$, we can construct a BAT $\mathcal{D}_{\mathcal{M}}$ such that

$$\mathcal{D}_{\mathcal{M}} \models \exists s. \text{Exec}(s) \wedge \text{Halt}(s) \text{ iff } \mathcal{M} \text{ halts}$$

where

- for each counter i , we add a nullary fluent f_i that stores the value of the counter i
- $\text{Next}(j)$ means j is the next instruction
- for each 2CM instruction j , add a corresponding action j that updates f_i and $\text{Next}(j)$

⁸ John McCarthy. *Situations, Actions, and Causal Laws*. Technical Report. Stanford University, 1963; Javier Pinto and Raymond Reiter. "Reasoning about Time in the Situation Calculus". In: *Ann. Math. Artif. Intell.* 14.2–4 (1995). DOI: 10.1007/BF01530822.

Existing Approaches: Real-Valued Bounded Fluents

- ▶ What if we *bound* the time, i.e., do not allow any values above a fixed bound b ?
- ▶ Time bounds do not help. Indeed:

Theorem

The time-bounded reachability problem for a real-valued BAT and a time bound $b \geq 2$ is undecidable.

Proof idea: Same reduction from the halting problem of 2CMs, except for the encoding of counter values:

- ▶ a counter value u is encoded by a fluent value 2^{1-u}
- ▶ an increment of i is encoded by an update of f_i to $\frac{f_i}{2}$
- ▶ a decrement of i is encoded by an update of f_i to $2f_i$

⇒ Allowing arbitrary real-valued fluents directly results in undecidability

Existing Approaches: Processes

Idea 2: Restrict continuous time to *processes*⁹ to obtain *timed BATs*

- ▶ Add timepoints to actions (and thus to situations)
- ▶ Actions may start and stop *processes*
- ▶ Formulas may only refer to the start time of processes
- ▶ **No real-valued fluents!**
- ▶ In the coffee domain:

$$P_{\text{brewing}}(p, t_{\text{start}}, \text{do}(a, s)) \equiv a = s\text{Brew}(p, t_{\text{start}}) \vee P_{\text{brewing}}(p, t_{\text{start}}, s) \wedge \forall t. a \neq e\text{Brew}(p, t)$$

$$\text{Strong}(p, \text{do}(a, s)) \equiv a = e\text{Brew}(p, t) \wedge P_{\text{brewing}}(p, t_{\text{start}}, s) \wedge t - t_{\text{start}} \geq 2$$

⁹Alberto Finzi and Fiora Pirri. "Representing Flexible Temporal Behaviors in the Situation Calculus". In: *IJCAI*. 2005.

Existing Approaches: Processes

Theorem

The reachability problem for timed BATs without real-valued functional fluents is undecidable.

Proof Idea: Reduction of the language emptiness problem for timed automata with additive clock constraints, which is undecidable¹⁰

- ▶ For each clock x , add a process $Reset_x$ that keeps track of the last clock reset
- ▶ Translate clock constraints to situation calculus formulas, e.g.,: $x + y = 1$ becomes

$$\mu_{x+y=1}(t, s) := \exists t_x, t_y. Reset_x(t_x, s) \wedge Reset_y(t_y, s) \wedge (t - t_x) + (t - t_y) = 1$$

¹⁰Béatrice Bérard and Catherine Dufourd. "Timed Automata and Additive Clock Constraints". In: *Information Processing Letters* 75.1 (July 31, 2000). doi: 10.1016/S0020-0190(00)00075-2.

How can we retain decidability?

- ▶ All *natural* ways of modeling dense time in the situation calculus fail:
 - Real-valued fluents allow arbitrary operations on the reals → undecidable
 - Upper-bounding the fluent values does not retain decidability
 - Restricting to *processes* still makes reachability undecidable

How can we retain decidability?

- ▶ All *natural* ways of modeling dense time in the situation calculus fail:
 - Real-valued fluents allow arbitrary operations on the reals → undecidable
 - Upper-bounding the fluent values does not retain decidability
 - Restricting to *processes* still makes reachability undecidable
- ▶ But: timed automata theory describes what we need to do:
 - Introduce **clocks** as a syntactically restricted form of real-valued fluents
 - All clock values increase uniformly and may only be reset to zero
 - Clock constraints may compare clock values only with integer constants

How can we retain decidability?

- ▶ All *natural* ways of modeling dense time in the situation calculus fail:
 - Real-valued fluents allow arbitrary operations on the reals → undecidable
 - Upper-bounding the fluent values does not retain decidability
 - Restricting to *processes* still makes reachability undecidable
- ▶ But: timed automata theory describes what we need to do:
 - Introduce **clocks** as a syntactically restricted form of real-valued fluents
 - All clock values increase uniformly and may only be reset to zero
 - Clock constraints may compare clock values only with integer constants
- ▶ Why does that work?
 - These restrictions allow us to apply *regionalization*
 - Regionalization defines an equivalence relation on clock values
 - We obtain a *time-abstract bisimulation*, a **finite** abstraction of the infinite tree of situations
 - ⇒ The finite structure permits a decidable procedure

How can we retain decidability?

- ▶ All *natural* ways of modeling dense time in the situation calculus fail:
 - Real-valued fluents allow arbitrary operations on the reals → undecidable
 - Upper-bounding the fluent values does not retain decidability
 - Restricting to *processes* still makes reachability undecidable
- ▶ But: timed automata theory describes what we need to do:
 - Introduce **clocks** as a syntactically restricted form of real-valued fluents
 - All clock values increase uniformly and may only be reset to zero
 - Clock constraints may compare clock values only with integer constants
- ▶ Why does that work?
 - These restrictions allow us to apply *regionalization*
 - Regionalization defines an equivalence relation on clock values
 - We obtain a *time-abstract bisimulation*, a **finite** abstraction of the infinite tree of situations
 - ⇒ The finite structure permits a decidable procedure
- ▶ Remainder of the talk: how can we integrate clocks into the situation calculus
 - ① *Clocked BATs*: integrate clocks into the situation calculus
 - ② *Regionalization*: abstract clock valuations into equivalent regions
 - ③ *Time-abstract Transition System*: a finite abstraction of the infinite tree of situations

Step 1: Clocked Basic Action Theories

- ▶ Add *clocks* as real-valued fluents
- ▶ Allow *clock comparisons* of the form $f(\vec{o}, s) \bowtie v$ and $v \bowtie v'$, where $v, v' \in \mathbb{N}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$
- ▶ A formula is called *clocked* if every sub-formula mentioning a term of sort *time* is a clock comparison
- ▶ A formula is *time-independent* if it does not mention any term of sort *time*

Step 1: Clocked Basic Action Theories

Definition

A BAT $\mathcal{D}$ is called a *clocked BAT* if:

- 1 For each real-valued fluent f , $\mathcal{D}_0$ contains an axiom $f(\vec{o}, s_0) = 0$.
- 2 For each real-valued fluent f , $\mathcal{D}_{ssa}$ contains an axiom of the form

$$f(\vec{o}, \text{do}(a, s)) = y \equiv \boxed{\exists t. a = \text{wait}(t) \wedge y = f(\vec{o}, s) + t} \quad \text{increase all clocks uniformly by } t$$
$$\vee \neg \exists t. a = \text{wait}(t) \wedge (\phi_f(\vec{o}, a, s) \wedge y = 0 \vee \neg \phi_f(\vec{o}, a, s) \wedge y = f(\vec{o}, s))$$

where $\phi_f(\vec{o}, a, s)$ is a **time-independent formula** uniform in s describing if the action a resets the value of $f(\vec{o})$.

- 3 For every relational fluent R , $\mathcal{D}_{ssa}$ contains an axiom of the form $R(\vec{o}, \text{do}(a, s)) \equiv \phi_R(\vec{o}, a, s)$ where $\phi_R(\vec{o}, a, s)$ is a **clocked formula** uniform in s .
- 4 Every precondition axiom in $\mathcal{D}_{poss}$ is of the form $\text{Poss}(A(\vec{o}), s) \equiv \phi_A(\vec{o}, s)$ such that $\phi_A(\vec{o}, s)$ is a **clocked formula** uniform in s .
- 5 Waiting for some non-negative amount of time is always possible: $\text{Poss}(\text{wait}(t), s) \equiv t \geq 0$.

Step 1: Clocked Basic Action Theories

Definition

A BAT $\mathcal{D}$ is called a *clocked BAT* if:

- 1 For each real-valued fluent f , $\mathcal{D}_0$ contains an axiom $f(\vec{o}, s_0) = 0$.
- 2 For each real-valued fluent f , $\mathcal{D}_{ssa}$ contains an axiom of the form

$$f(\vec{o}, \text{do}(a, s)) = y \equiv \exists t. a = \text{wait}(t) \wedge y = f(\vec{o}, s) + t$$

*reset $f(\vec{o})$ to 0 if $\phi_f(\vec{o}, a)$ is true,
leave unchanged otherwise*

$$\vee \neg \exists t. a = \text{wait}(t) \wedge (\phi_f(\vec{o}, a, s) \wedge y = 0 \vee \neg \phi_f(\vec{o}, a, s) \wedge y = f(\vec{o}, s))$$

where $\phi_f(\vec{o}, a, s)$ is a **time-independent formula** uniform in s describing if the action a resets the value of $f(\vec{o})$.

- 3 For every relational fluent R , $\mathcal{D}_{ssa}$ contains an axiom of the form $R(\vec{o}, \text{do}(a, s)) \equiv \phi_R(\vec{o}, a, s)$ where $\phi_R(\vec{o}, a, s)$ is a **clocked formula** uniform in s .
- 4 Every precondition axiom in $\mathcal{D}_{poss}$ is of the form $\text{Poss}(A(\vec{o}), s) \equiv \phi_A(\vec{o}, s)$ such that $\phi_A(\vec{o}, s)$ is a **clocked formula** uniform in s .
- 5 Waiting for some non-negative amount of time is always possible: $\text{Poss}(\text{wait}(t), s) \equiv t \geq 0$.

Step 1: Clocked Basic Action Theories

- ▶ In a *clocked* BAT, real-valued fluents function like *clocks* in timed automata:
 - $wait(t)$ increments all clock values by t
 - All other actions leave a clock value unchanged or reset it to zero
 - Clocks may have arguments
- ▶ We also write C for the set of ground clocks, i.e., the ground function fluents
- ▶ A *clock valuation* v_σ in situation σ is the value of all ground clock fluents:

$$v_\sigma = \{(c, v) \mid c \in C, \mathcal{D} \models c[\sigma] = v\}$$

Example (I)

- Initially, the robot is doing nothing, nothing is filled, the coffee is not strong, and all clocks are initialized to zero. Also, only Mug_1 should be filled with strong coffee:

$$\neg Brew(p, S_0), \neg Pour(p, m, S_0), \neg IsFull(o, S_0), \neg Strong(o, S_0), \\ c_{brew}(p, S_0) = 0, c_{glob}(S_0) = 0, WantStrong(o) \equiv o = Mug_1$$

- The robot may start a durative action if it is not doing anything and end it if it has started it before. Brewing takes at least 1 time unit. In order to pour the coffee, there must be coffee in the pot:

$$Poss(sBrew(p), s) \equiv \neg Brew(p, s) \wedge \neg \exists m. Pour(p, m, s)$$

$$Poss(eBrew(p), s) \equiv Brew(p, s) \wedge c_{brew}(p, s) \geq 1$$

$$Poss(sPour(p, m), s) \equiv IsFull(p, s) \wedge \neg \exists p', m'. Pour(p', m', s) \wedge \neg Brew(p, s)$$

$$Poss(ePour(p, m), s) \equiv Pour(p, m, s)$$

$$Poss(wait(t), s) \equiv t \geq 0$$

Example (II)

- The SSAs state that brew fills a pot with coffee and makes it strong if it took at least 2 time units:

$$IsFull(o, do(a, s)) \equiv a = eBrew(o) \vee \exists p. a = ePour(p, o) \vee IsFull(o, s)$$

$$Strong(o, do(a, s)) \equiv a = eBrew(o) \wedge c_{brew}(o, s) \geq 2$$

$$\vee \exists p. a = ePour(p, o) \wedge Strong(p, s) \vee Strong(o, s)$$

$$Brew(p, do(a, s)) \equiv a = sBrew(p) \vee Brew(p, s) \wedge a \neq eBrew(p)$$

$$Pour(p, m, do(a, s)) \equiv a = sPour(p, m) \vee Pour(p, m, s) \wedge a \neq ePour(p, m)$$

$$c_{brew}(p, do(a, s)) = y \equiv \exists t(a = wait(t) \wedge y = c_{brew}(p, s) + t) \vee (\neg \exists t. a = wait(t)$$

$$\wedge (a = sBrew(p) \wedge y = 0 \vee a \neq sBrew(p) \wedge y = c_{brew}(p, s)))$$

$$c_{glob}(do(a, s)) = y \equiv \exists t. a = wait(t) \wedge y = c_{glob}(s) + t \vee \neg \exists t. a = wait(t) \wedge y = c_{glob}(s)$$

- There are no other objects or actions:

$$\forall o(o = Pot_1 \vee o = Pot_2 \vee o = Mug_1 \vee o = Mug_2)$$

Step 2: Regionalization

- ▶ Clocks may have infinitely many values and wait produces infinitely many successors (one for each $t \in \mathbb{R}$)
 - ▶ However:
 - All clock values progress uniformly
 - Clocks may only be reset to zero
 - Clock values may only be compared against integers
 - There is a maximal constant $\mathcal{K} \in \mathbb{N}$ in all comparisons
- Use *regionalization*:¹¹ a finite abstraction of all possible clock valuations

¹¹Rajeev Alur and David L. Dill. "A Theory of Timed Automata". In: *Theoretical Computer Science* 126.2 (Apr. 25, 1994). doi: 10.1016/0304-3975(94)90010-8.

Step 2: Regionalization

Definition (Clock Regions)

Given a maximal constant $\mathcal{K}$, let $V = [0, \mathcal{K}] \cup \{\top\}$. We define the *region equivalence* as the equivalence relation $\sim_{\mathcal{K}}$ on V such that $u \sim_{\mathcal{K}} v$ if

- ▶ $u = v = \top$, or
- ▶ $u, v \neq \top$, $\lceil u \rceil = \lceil v \rceil$, and $\lfloor u \rfloor = \lfloor v \rfloor$.

A *region* is an equivalence class of $\sim_{\mathcal{K}}$.

Definition (Region Equivalence)

Let v, v' be two clock valuations over C , then v and v' are region-equivalent, written $v \cong_{\mathcal{K}} v'$ iff

- ① for every $c \in C$, $v(c) \sim_{\mathcal{K}} v'(c)$, and
- ② for every $c, c' \in C$, $\text{fract}(v(c)) \leq \text{fract}(v(c'))$ iff $\text{fract}(v'(c)) \leq \text{fract}(v'(c'))$.

Step 2: Regionalization

Definition (Clock Regions)

Given a maximal constant $\mathcal{K}$, let $V = [0, \mathcal{K}] \cup \{\top\}$. We define the *region equivalence* as the equivalence relation $\sim_{\mathcal{K}}$ on V such that $u \sim_{\mathcal{K}} v$ if

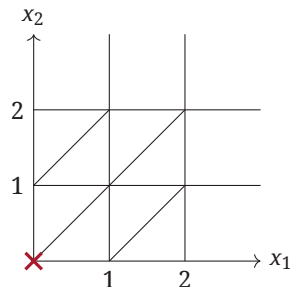
- ▶ $u = v = \top$, or
- ▶ $u, v \neq \top$, $\lceil u \rceil = \lceil v \rceil$, and $\lfloor u \rfloor = \lfloor v \rfloor$.

A *region* is an equivalence class of $\sim_{\mathcal{K}}$.

Definition (Region Equivalence)

Let v, v' be two clock valuations over C , then v and v' are region-equivalent, written $v \cong_{\mathcal{K}} v'$ iff

- 1 for every $c \in C$, $v(c) \sim_{\mathcal{K}} v'(c)$, and
- 2 for every $c, c' \in C$, $\text{fract}(v(c)) \leq \text{fract}(v(c'))$ iff $\text{fract}(v'(c)) \leq \text{fract}(v'(c'))$.



$$x_1 = 0 \wedge x_2 = 0$$

Step 2: Regionalization

Definition (Clock Regions)

Given a maximal constant $\mathcal{K}$, let $V = [0, \mathcal{K}] \cup \{\top\}$. We define the *region equivalence* as the equivalence relation $\sim_{\mathcal{K}}$ on V such that $u \sim_{\mathcal{K}} v$ if

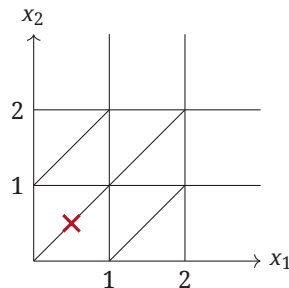
- ▶ $u = v = \top$, or
- ▶ $u, v \neq \top$, $\lceil u \rceil = \lceil v \rceil$, and $\lfloor u \rfloor = \lfloor v \rfloor$.

A *region* is an equivalence class of $\sim_{\mathcal{K}}$.

Definition (Region Equivalence)

Let v, v' be two clock valuations over C , then v and v' are region-equivalent, written $v \cong_{\mathcal{K}} v'$ iff

- 1 for every $c \in C$, $v(c) \sim_{\mathcal{K}} v'(c)$, and
- 2 for every $c, c' \in C$, $\text{fract}(v(c)) \leq \text{fract}(v(c'))$ iff $\text{fract}(v'(c)) \leq \text{fract}(v'(c'))$.



$$x_1 > 0 \wedge x_1 < 1 \wedge x_2 > 0 \wedge x_2 < 1 \wedge x_1 = x_2$$

Step 2: Regionalization

Definition (Clock Regions)

Given a maximal constant $\mathcal{K}$, let $V = [0, \mathcal{K}] \cup \{\top\}$. We define the *region equivalence* as the equivalence relation $\sim_{\mathcal{K}}$ on V such that $u \sim_{\mathcal{K}} v$ if

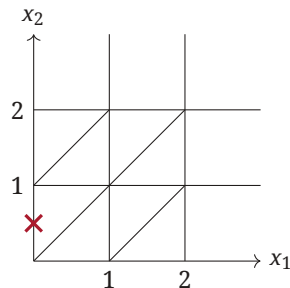
- ▶ $u = v = \top$, or
- ▶ $u, v \neq \top$, $\lceil u \rceil = \lceil v \rceil$, and $\lfloor u \rfloor = \lfloor v \rfloor$.

A *region* is an equivalence class of $\sim_{\mathcal{K}}$.

Definition (Region Equivalence)

Let v, v' be two clock valuations over C , then v and v' are region-equivalent, written $v \cong_{\mathcal{K}} v'$ iff

- 1 for every $c \in C$, $v(c) \sim_{\mathcal{K}} v'(c)$, and
- 2 for every $c, c' \in C$, $\text{fract}(v(c)) \leq \text{fract}(v(c'))$ iff $\text{fract}(v'(c)) \leq \text{fract}(v'(c'))$.



$$x_1 = 0 \wedge x_2 > 0 \wedge x_2 < 1$$

Step 2: Regionalization

Definition (Clock Regions)

Given a maximal constant $\mathcal{K}$, let $V = [0, \mathcal{K}] \cup \{\top\}$. We define the *region equivalence* as the equivalence relation $\sim_{\mathcal{K}}$ on V such that $u \sim_{\mathcal{K}} v$ if

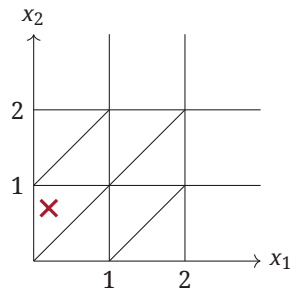
- ▶ $u = v = \top$, or
- ▶ $u, v \neq \top$, $\lceil u \rceil = \lceil v \rceil$, and $\lfloor u \rfloor = \lfloor v \rfloor$.

A *region* is an equivalence class of $\sim_{\mathcal{K}}$.

Definition (Region Equivalence)

Let v, v' be two clock valuations over C , then v and v' are region-equivalent, written $v \cong_{\mathcal{K}} v'$ iff

- 1 for every $c \in C$, $v(c) \sim_{\mathcal{K}} v'(c)$, and
- 2 for every $c, c' \in C$, $\text{fract}(v(c)) \leq \text{fract}(v(c'))$ iff $\text{fract}(v'(c)) \leq \text{fract}(v'(c'))$.



$$x_1 > 0 \wedge x_1 < 1 \wedge x_2 > 0 \wedge x_2 < 1 \wedge x_1 < x_2$$

Step 2: Regionalization

Definition (Clock Regions)

Given a maximal constant $\mathcal{K}$, let $V = [0, \mathcal{K}] \cup \{\top\}$. We define the *region equivalence* as the equivalence relation $\sim_{\mathcal{K}}$ on V such that $u \sim_{\mathcal{K}} v$ if

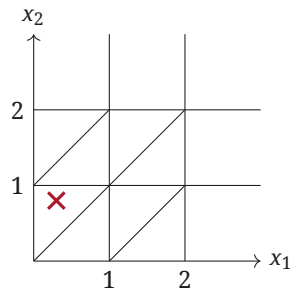
- ▶ $u = v = \top$, or
- ▶ $u, v \neq \top$, $\lceil u \rceil = \lceil v \rceil$, and $\lfloor u \rfloor = \lfloor v \rfloor$.

A *region* is an equivalence class of $\sim_{\mathcal{K}}$.

Definition (Region Equivalence)

Let v, v' be two clock valuations over C , then v and v' are region-equivalent, written $v \cong_{\mathcal{K}} v'$ iff

- 1 for every $c \in C$, $v(c) \sim_{\mathcal{K}} v'(c)$, and
- 2 for every $c, c' \in C$, $\text{fract}(v(c)) \leq \text{fract}(v(c'))$ iff $\text{fract}(v'(c)) \leq \text{fract}(v'(c'))$.



$$x_1 > 0 \wedge x_1 < 1 \wedge x_2 > 0 \wedge x_2 < 1 \wedge x_1 < x_2$$

Step 2: Regionalization

Definition (Clock Regions)

Given a maximal constant $\mathcal{K}$, let $V = [0, \mathcal{K}] \cup \{\top\}$. We define the *region equivalence* as the equivalence relation $\sim_{\mathcal{K}}$ on V such that $u \sim_{\mathcal{K}} v$ if

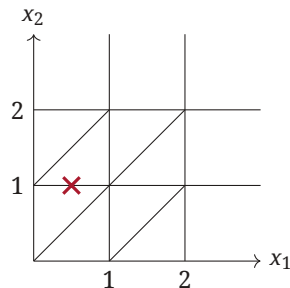
- ▶ $u = v = \top$, or
- ▶ $u, v \neq \top$, $\lceil u \rceil = \lceil v \rceil$, and $\lfloor u \rfloor = \lfloor v \rfloor$.

A *region* is an equivalence class of $\sim_{\mathcal{K}}$.

Definition (Region Equivalence)

Let v, v' be two clock valuations over C , then v and v' are region-equivalent, written $v \cong_{\mathcal{K}} v'$ iff

- 1 for every $c \in C$, $v(c) \sim_{\mathcal{K}} v'(c)$, and
- 2 for every $c, c' \in C$, $\text{fract}(v(c)) \leq \text{fract}(v(c'))$ iff $\text{fract}(v'(c)) \leq \text{fract}(v'(c'))$.



$$x_1 > 0 \wedge x_1 < 1 \wedge x_2 = 1$$

Step 2: Regionalization

Definition (Clock Regions)

Given a maximal constant $\mathcal{K}$, let $V = [0, \mathcal{K}] \cup \{\top\}$. We define the *region equivalence* as the equivalence relation $\sim_{\mathcal{K}}$ on V such that $u \sim_{\mathcal{K}} v$ if

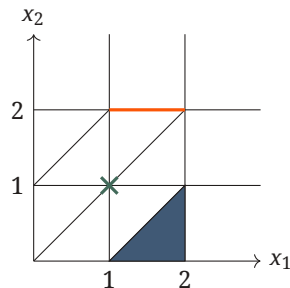
- ▶ $u = v = \top$, or
- ▶ $u, v \neq \top$, $\lceil u \rceil = \lceil v \rceil$, and $\lfloor u \rfloor = \lfloor v \rfloor$.

A *region* is an equivalence class of $\sim_{\mathcal{K}}$.

Definition (Region Equivalence)

Let v, v' be two clock valuations over C , then v and v' are region-equivalent, written $v \cong_{\mathcal{K}} v'$ iff

- 1 for every $c \in C$, $v(c) \sim_{\mathcal{K}} v'(c)$, and
- 2 for every $c, c' \in C$, $\text{fract}(v(c)) \leq \text{fract}(v(c'))$ iff $\text{fract}(v'(c)) \leq \text{fract}(v'(c'))$.



Step 3: Time-Abstract Transition System

Definition (Situation Equivalence)

Let $\approx_{\mathcal{D}\mathcal{K}} \subseteq \mathcal{S} \times \mathcal{S}$ be an equivalence relation such that $\sigma_1 \approx_{\mathcal{D}\mathcal{K}} \sigma_2$ iff

- ① both situations agree on all fluents, i.e., for every fluent $R \in \mathcal{F}$, $\mathcal{D} \models R(\vec{\rho}, \sigma_1)$ iff $\mathcal{D} \models R(\vec{\rho}, \sigma_2)$,
- ② the two clock valuations are region-equivalent, i.e., $v_{\sigma_1} \cong_{\mathcal{K}} v_{\sigma_2}$.

Definition (Time-abstract Transition System)

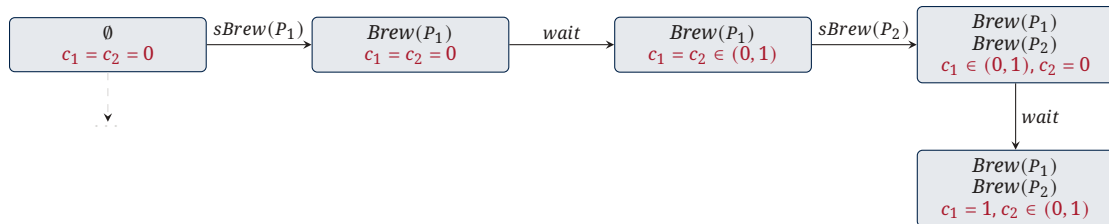
Let $\mathcal{D}$ be a clocked BAT and $\mathcal{K}$ the maximal constant appearing in $\mathcal{D}$.

We define the TS $\mathcal{T}_{\mathcal{D}} = (Q, q_0, \rightarrow)$ as follows:

- ▶ The set of states Q consists of equivalence classes of $\approx_{\mathcal{D}\mathcal{K}}$, i.e., $Q \subseteq \mathcal{S} / \approx_{\mathcal{D}\mathcal{K}}$;
- ▶ The initial state is the equivalence class $q_0 = [S_0]_{\approx_{\mathcal{D}\mathcal{K}}}$;
- ▶ $[\sigma]_{\approx_{\mathcal{D}\mathcal{K}}} \rightarrow [\sigma']_{\approx_{\mathcal{D}\mathcal{K}}}$ if σ' is an executable successor situation of σ , i.e., if $\sigma' = \text{do}(\alpha, \sigma)$ and $\mathcal{D} \models \text{Exec}(\sigma')$.

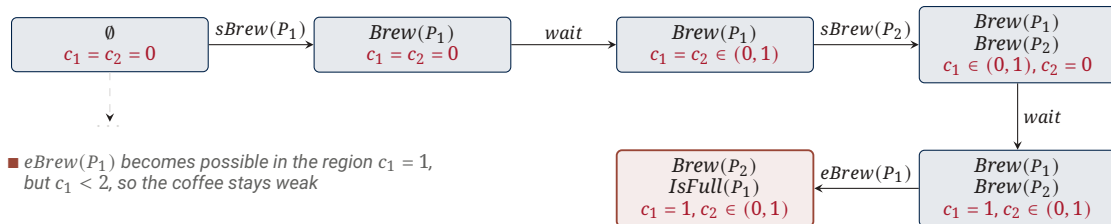
Example

Consider the coffee domain with two pots P_1, P_2 and one mug M :



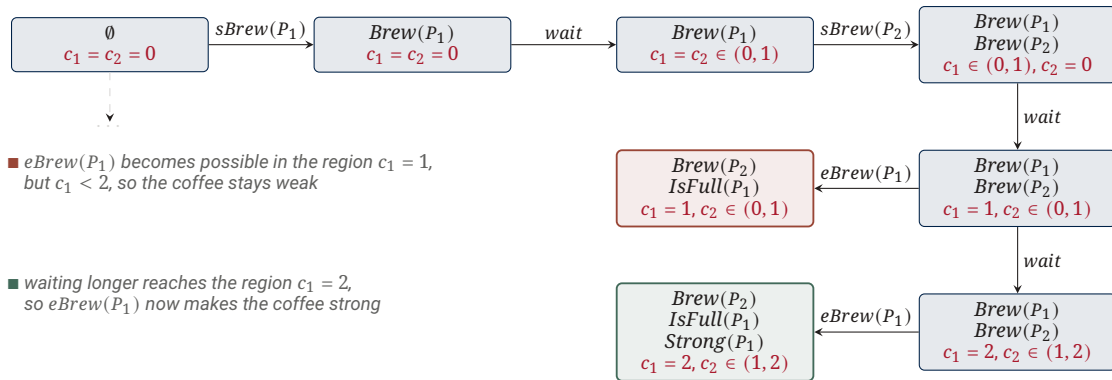
Example

Consider the coffee domain with two pots P_1, P_2 and one mug M :



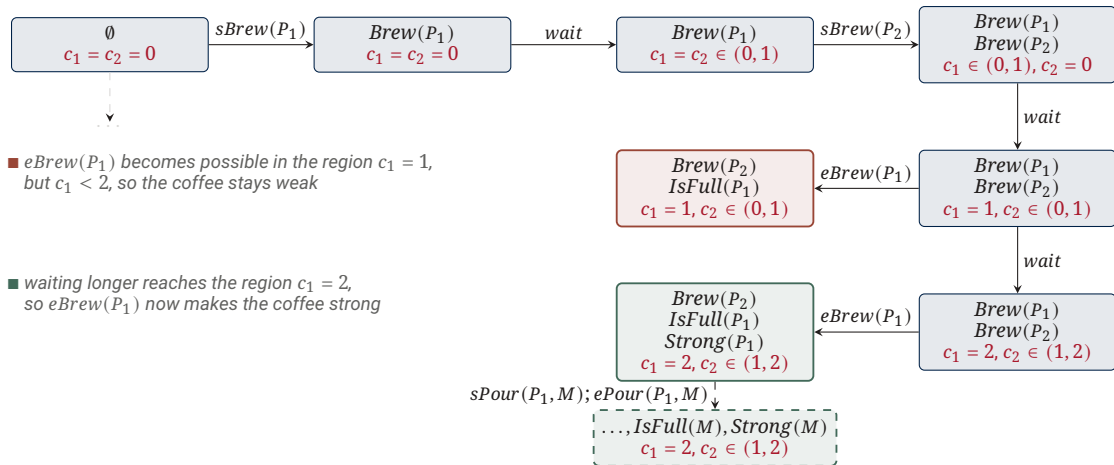
Example

Consider the coffee domain with two pots P_1, P_2 and one mug M :



Example

Consider the coffee domain with two pots P_1, P_2 and one mug M :



Decidability

We can make the following observations:

- 1 $[\sigma]_{\approx_{\mathcal{D}\mathcal{K}}}$ is reachable in $\mathcal{T}_{\mathcal{D}}$ iff there is some $\sigma' \in [\sigma]_{\approx_{\mathcal{D}\mathcal{K}}}$ with $\mathcal{D} \models \text{Exec}(\sigma')$.
- 2 If $\sigma_1 \approx_{\mathcal{D}\mathcal{K}} \sigma_2$, then for every clocked formula ϕ : $\mathcal{D} \models \phi[\sigma_1]$ iff $\mathcal{D} \models \phi[\sigma_2]$.
- 3 The time-abstract transition system $\mathcal{T}_{\mathcal{D}}$ has finitely many states.

Combining these, we obtain:

Theorem

Reachability for clocked BATs is decidable.

Reachability: Examples

Consider the coffee domain with one pot and two mugs, and $\mathcal{D}_0 \models \text{WantStrong}(o) \equiv o = \text{Mug}_1$, i.e., only Mug_1 should be filled with strong coffee. We can check the following reachability queries:

- Can we fill all mugs and pots? $\mathcal{D} \models \exists s. \text{Exec}(s) \wedge \forall o. \text{IsFull}(o, s)$

Yes, for example:

$$\sigma_1 = \text{do}([s\text{Brew}(\text{Pot}); \text{wait}(1); e\text{Brew}(\text{Pot}); s\text{Pour}(\text{Pot}, \text{Mug}_1); e\text{Pour}(\text{Pot}, \text{Mug}_1); \\ s\text{Pour}(\text{Pot}, \text{Mug}_2); e\text{Pour}(\text{Pot}, \text{Mug}_2)], S_0)$$

- Can we fill all mugs with the right type of coffee? $\mathcal{D} \models \exists s. \text{Exec}(s) \wedge \forall o. \text{Strong}(o, s) \equiv \text{WantStrong}(o)$

Yes, for example:

$$\sigma_2 = \text{do}([s\text{Brew}(\text{Pot}); s\text{Brew}(\text{Mug}_1); \text{wait}(1); e\text{Brew}(\text{Pot}); \text{wait}(2); e\text{Brew}(\text{Mug}_1); \\ s\text{Pour}(\text{Pot}, \text{Mug}_2); \text{wait}(1); e\text{Pour}(\text{Pot}, \text{Mug}_2)], S_0)$$

Realizability

Program realization for GoLOG programs:

- ▶ Given: a BAT $\mathcal{D}$ and a program δ .
- ▶ Find an action sequence $\langle \alpha_1, \dots, \alpha_n \rangle$ that is a *realization* of δ , i.e.,:
 - 1 Each action must be possible when it occurs: $\mathcal{D} \models \text{Poss}(\alpha_{i+1}, \text{do}(\alpha_i, \text{do}(\dots, S_0)))$
 - 2 The program must end in a *final configuration* (i.e., no remaining program)

Theorem

The realization problem for a clocked program $\Delta = (\mathcal{D}, \delta)$ is decidable.

Program Realization: Example

- Consider the following program (again two mugs, one pot):

$$(sBrew(Pot); \exists m (WantStrong(m)) \equiv c_{brew}(Pot) \geq 2?; eBrew(Pot); \\ \mathbf{while} (\exists m \neg IsFull(m)) \mathbf{do} \pi m. sPour(Pot, m); ePour(Pot, m); \mathbf{done}) \parallel wait()^*$$

The program is realizable, e.g., with

$$sBrew(Pot); wait(2); eBrew(Pot); sPour(Pot, Mug_1); \\ ePour(Pot, Mug_1); sPour(Pot, Mug_2); ePour(Pot, Mug_2);$$

Program Realization: Example

- Consider the following program (again two mugs, one pot):

$$(sBrew(Pot); \exists m (WantStrong(m)) \equiv c_{brew}(Pot) \geq 2?; eBrew(Pot); \\ \mathbf{while} (\exists m \neg IsFull(m)) \mathbf{do} \pi m. sPour(Pot, m); ePour(Pot, m); \mathbf{done}) \parallel wait()^*$$

The program is realizable, e.g., with

$$sBrew(Pot); wait(2); eBrew(Pot); sPour(Pot, Mug_1); \\ ePour(Pot, Mug_1); sPour(Pot, Mug_2); ePour(Pot, Mug_2);$$

- Consider the following variant:

$$(sBrew(Pot); \exists m (WantStrong(m)) \equiv c_{brew}(Pot) \geq 2?; eBrew(Pot); \\ \mathbf{while} (\exists m \neg IsFull(m)) \mathbf{do} \pi m. sPour(Pot, m); ePour(Pot, m); \mathbf{done}) \parallel wait()^*; \\ \forall m. (Strong(m) \equiv WantStrong(m))?$$

This program is **not** realizable. Why?

Conclusion

- ▶ We considered several **models of time** in the situation calculus
- ▶ Existing approaches all result in **undecidability**
- ▶ Solution: Add *clocks* as real-valued fluents with syntactic restrictions, akin to timed automata
- **Reachability** and **program realization** are decidable

Limitations and future work:

- ▶ So far: restriction to a fixed and finite set of objects → not more expressive than timed automata
- ⇒ Extend to more expressive decidable fragments, e.g., *bounded action theories*

Appendix

Bibliography (I)

-  Alur, Rajeev and David L. Dill. “A Theory of Timed Automata”. In: *Theoretical Computer Science* 126.2 (Apr. 25, 1994). DOI: 10.1016/0304-3975(94)90010-8.
-  Bérard, Béatrice and Catherine Dufourd. “Timed Automata and Additive Clock Constraints”. In: *Information Processing Letters* 75.1 (July 31, 2000). DOI: 10.1016/S0020-0190(00)00075-2.
-  Claßen, Jens. “Planning and Verification in the Agent Language Golog”. PhD thesis. Aachen, Germany: RWTH Aachen University, 2013.
-  —. “Symbolic Verification of Golog Programs with First-Order BDDs”. In: *KR*. 2018.
-  De Giacomo, Giuseppe, Yves Lespérance, and Fabio Patrizi. “Bounded Situation Calculus Action Theories”. In: *AIJ* 237 (Aug. 1, 2016). DOI: 10.1016/j.artint.2016.04.006.
-  De Giacomo, Giuseppe et al. “Reactive Synthesis for Golog Specifications in the Propositional Situation Calculus”. In: *KR*. 2026.
-  Finzi, Alberto and Fiora Pirri. “Representing Flexible Temporal Behaviors in the Situation Calculus”. In: *IJCAI*. 2005.
-  Hofmann, Till and Jens Claßen. “LTLf Synthesis on First-Order Agent Programs in Nondeterministic Environments”. In: *AAAI*. 2025. DOI: 10.1609/aaai.v39i14.33642.
-  McCarthy, John. *Situations, Actions, and Causal Laws*. Technical Report. Stanford University, 1963.

Bibliography (II)



McCarthy, John and Patrick J. Hayes. “Some Philosophical Problems from the Standpoint of Artificial Intelligence”. In: *Mach. Intell.* 4 (1969).



Pinto, Javier and Raymond Reiter. “Reasoning about Time in the Situation Calculus”. In: *Ann. Math. Artif. Intell.* 14.2–4 (1995). DOI: [10.1007/BF01530822](https://doi.org/10.1007/BF01530822).



Reiter, Raymond. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.



Zarrieß, Benjamin and Jens Claßen. “Decidable Verification of Golog Programs over Non-Local Effect Actions”. In: *AAAI*. 2016. DOI: [10.1609/aaai.v30i1.10109](https://doi.org/10.1609/aaai.v30i1.10109).

Till Hofmann, Stefan Schupp, Gerhard Lakemeyer

TIME'26, Cork, Ireland, September 3, 2026