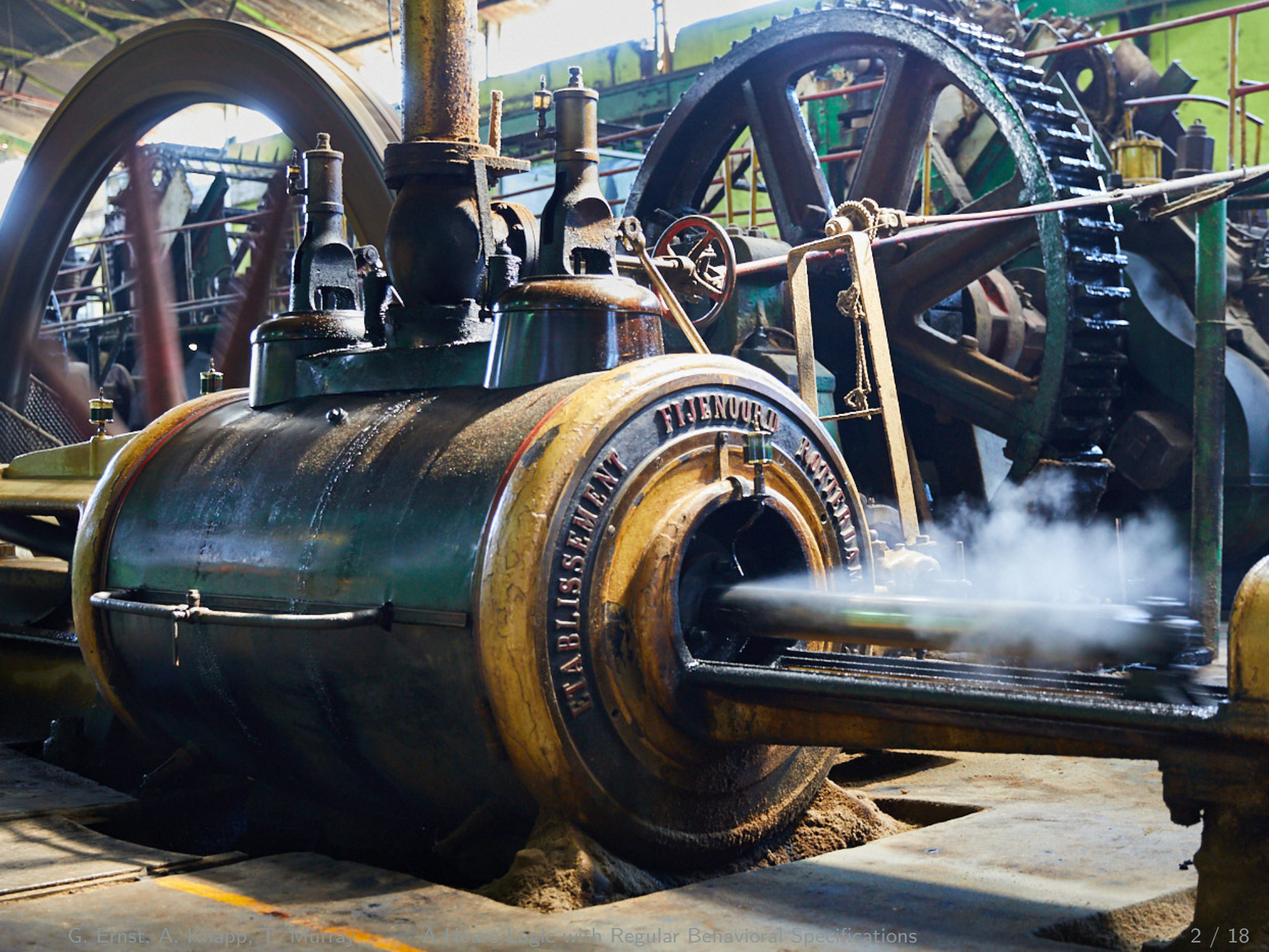
A Hoare Logic with Regular Behavioral Specifications

Gidon Ernst — LMU Munich
Alexander Knapp — University of Augsburg
Toby Murray — University of Melbourne

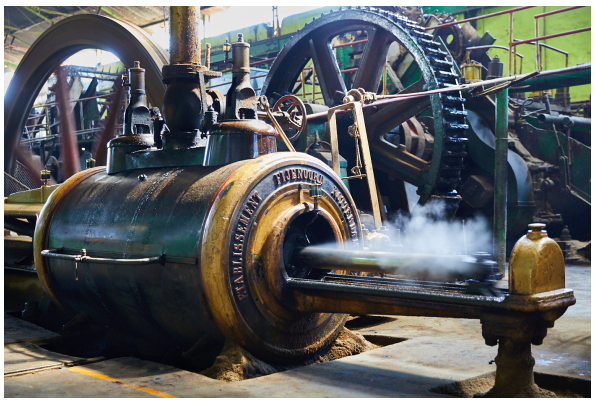
ISoLA 2022, Rhodes

https://bitbucket.org/covern/secc

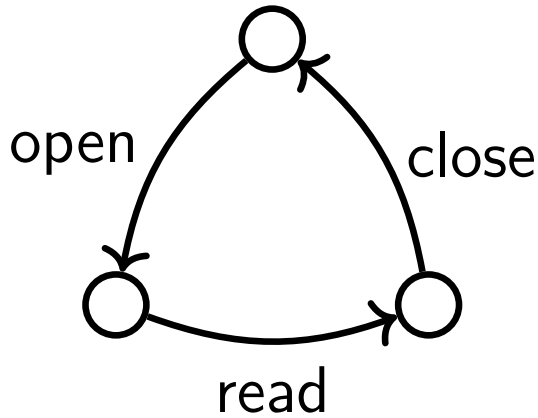

ETABLISSEMENT
FIJENOORD

Motivation

System Code



Interfaces



Hoare-like

- contracts
- data invariants

abstraction gap

behavioral

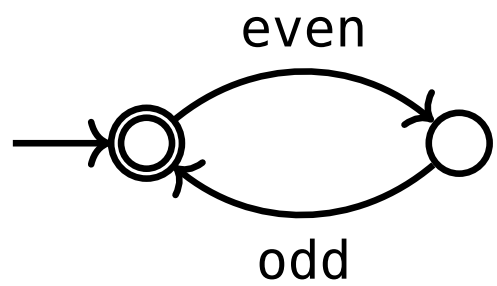
- events, traces
- protocols

Example

```
void even_odd()

{
  bool b = false;

  while(*)


  {
    if(!b) print("even");
    else   print("odd");
    b = !b;
  }
}
```

Behavioral Abstractions

- as a finite automaton



- as a regular language

$$(\mathsf{even}{\cdot}\mathsf{odd})^*$$

- need to keep track of states

$$tr(\texttt{while}\ t\ \texttt{do}\ b) \subsetneq tr(b)^*$$

Annotated Example in SecC

```
void even_odd()
    _(trace (even odd)*)
{
  bool b = false;

  while(*)
    _(trace (even odd)*      if  b)
    _(trace (even odd)* even if !b)
  {
    if(!b) print("even");
    else   print("odd");
    b = !b;
  }
}
```

```
even_odd ...   failed
  event trace
    (even odd)* even
                ^^^^
             unmatched

residual specification
  odd (even odd)*

path condition
  b == true
```

https://bitbucket.org/covern/secc/src/master/examples/even_odd.c

Context and Motivation

Goal: verify low-level code + high-level behavior

Existing approaches

- history- vs. future-based [Blom+'15, Jacobs'20]
- separation logic + process algebras [Oortwijn+'20]
- session types e.g. [Hüttel+'16]

complex or not fitting well with Hoare logic

Our Interests:

- confidence in the implementation in SecC
  (spoiler: our first attempt was wrong)
- work out the principles of trace specifications
- understand design and trade-offs

Contribution

Approach: Hoare-logic with judgements $\{ P : U \}\ c\ \{ Q : V \}$

- Trace specifications U and V over regular expressions u_i

$$U, V ::= \quad u_1 \texttt{ if } \phi_1 \quad \texttt{+} \quad \cdots \quad \texttt{+} \quad u_n \texttt{ if } \phi_n$$

- Specification commands that model occurrence of events

$$c ::= \texttt{emit } U \mid \cdots$$

Results

- Clean, simple extension of Hoare logic
- ✓ Sound (via Isabelle) and complete (sketch) proof rules
- Tool implementation in SecC
- Case studies: Casino, Regex matcher

`emit`: History vs. Future

Implemented: U captures the history seen so far

$$\frac{}{\{\, P \mathbin{:} U \,\}\ \texttt{emit}\ V\ \{\, P \mathbin{:} U \cdot V \,\}}\ \textsc{Emit}^{\rightarrow}$$

- adequate for *forward* symbolic execution

Alternative: U specifies what is expected

$$\frac{}{\{\, P \mathbin{:} V \cdot U \,\}\ \texttt{emit}\ V\ \{\, P \mathbin{:} U \,\}}\ \textsc{Emit}^{\leftarrow}$$

- *backwards* reasoning—no best forward transformer
  $\{\, P \mathbin{:} (a \cdot a) + (b \cdot b) \,\}\ \texttt{emit}\ (a + b)\ \{\, P \mathbin{:} \textbf{???} \,\}$
- but can capture correctness for infinite runs

Sequential Composition

Program anntotations mention just *the* trace

```
void even_odd() _(trace (even odd)*)
```

Trace specifications are *state-dependent*

```
b = false;
_(emit even if b + odd if !b)      U0
b = true;
_(emit even if b + odd if !b)      U1
```

Result is *not* $U_0 \cdot U_1$ but $(U_0[\texttt{b} \mapsto \texttt{false}]) \cdot (U_1[\texttt{b} \mapsto \texttt{true}])$

Proof rules are symmetric to achieve modularity

$$\frac{\{\, P : U \,\}\ c_1\ \{\, Q : V \,\} \qquad \{\, Q : V \,\}\ c_2\ \{\, R : W \,\}}{\{\, P : U \,\}\ c_1; c_2\ \{\, R : W \,\}}\ \textsc{Seq}$$

A “temporal frame rule”

Program execution is independent of trace prefix W

$$\frac{P \wedge Q\sigma \Longrightarrow W \equiv W\sigma \qquad \{\, P : U \,\}\ c\ \{\, R : V \,\}}{\{\, P : W \cdot U \,\}\ c\ \{\, Q : W \cdot V \,\}}\ \textsc{Frame}^{\rightarrow}$$

- condition: description W of prefix trace is preserved by c where $\sigma = [\vec{x} \mapsto \vec{x}']$ renames $\vec{x} = mod(c)$
- language equivalence $W \equiv W\sigma$ wrt. states P and $Q\sigma$

Future-based: W is postfix
Analogy: spatial frame rule in separation logic

Consequence Rule

History-based:

$$\frac{\{\, P_2 \mid U_2 \,\}\ c\ \{\, Q_2 \mid V_2 \,\}}{\{\, P_1 \mid U_1 \,\}\ c\ \{\, Q_1 \mid V_1 \,\}}\ \textsc{Conseq}^{\rightarrow}$$

where

- $P_1 \Longrightarrow P_2 \wedge (U_1 \sqsubseteq U_2)$
- $Q_2 \Longrightarrow Q_1 \wedge (V_2 \sqsubseteq V_1)$

Future-based:

- swap $\sqsubseteq$ to $\sqsupseteq$

While Loops

$$\frac{\{\, t \wedge I \,\vdots\, U \,\}\ c\ \{\, I \,\vdots\, U \,\}}{\{\, I \,\vdots\, U \,\}\ \texttt{while}\ t\ \texttt{do}\ c\ \{\, \neg t \wedge I \,\vdots\, U \,\}}\ \textsc{While}$$

- occurrences of U refer to different states

Side-conditions from the example

when `!b` $\quad (\mathsf{even}\cdot\mathsf{odd})^*\cdot\underline{\mathsf{even}} \sqsubseteq (\mathsf{even}\cdot\mathsf{odd})^*\cdot\mathsf{even}$

when `b` $\quad (\mathsf{even}\cdot\mathsf{odd})^*\cdot\mathsf{even}\cdot\underline{\mathsf{odd}} \sqsubseteq (\mathsf{even}\cdot\mathsf{odd})^*$

Contribution

Approach: Hoare-logic with judgements $\{ P : U \}\ c\ \{ Q : V \}$

- Trace specifications U and V over regular expressions u_i

$$U, V ::= \quad u_1 \texttt{ if } \phi_1 \quad \texttt{+} \quad \cdots \quad \texttt{+} \quad u_n \texttt{ if } \phi_n$$

- Specification commands that model occurrence of events

$$c ::= \texttt{emit } U \mid \cdots$$

Results

- Clean, simple extension of Hoare logic
- ✓ Sound (via Isabelle) and complete (sketch) proof rules
- Tool implementation in SecC
- Case studies: Casino, Regex matcher

SMT-decidable side-conditions

Query $P \Longrightarrow U \sqsubseteq V$ iff $\underbrace{U_P \sqsubseteq V_P}_{\text{plain regular}}$

where $(u \texttt{ if } \phi)_P = \begin{cases} u, & \text{if } \phi \wedge P \texttt{ sat} \\ \varnothing, & \text{otherwise} \end{cases}$

Algorithmic check $u \sqsubseteq v$ via Brzozowski derivative

Note: only place where regularity of traces is relevant

Contribution

Approach: Hoare-logic with judgements $\{ P : U \}\ c\ \{ Q : V \}$

- Trace specifications U and V over regular expressions u_i

$$U, V ::= \quad u_1 \texttt{ if } \phi_1 \quad \texttt{+} \quad \cdots \quad \texttt{+} \quad u_n \texttt{ if } \phi_n$$

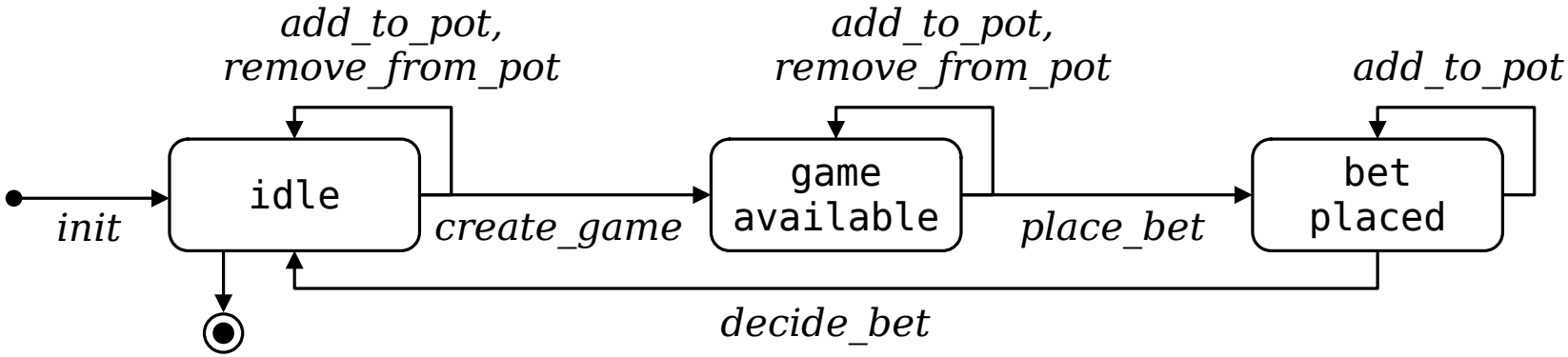
- Specification commands that model occurrence of events

$$c ::= \texttt{emit } U \mid \cdots$$

Results

- Clean, simple extension of Hoare logic
- ✓ Sound (via Isabelle) and complete (sketch) proof rules
- Tool implementation in SecC
- Case studies: Casino, Regex matcher

Casino




```
int decide_bet(struct address *self, struct address *sender,
               struct casino *casino, int secret)
  _(maintains game(...))
  _(requires  sender == _operator)
  _(requires  _hash == _cryptohash(secret))
  _(ensures   result ==> payment(self, _player; 2*_bet))
  _(trace  player_wins if result  +  casino_wins if !result)
```

https://bitbucket.org/covern/secc/src/master/examples/case-studies/casino.c

Conclusion

Thanks: everybody for participating in the discussions!

Results

- Clean, simple extension of Hoare logic
- Sound (via Isabelle) and complete (sketch) proof rules
- Tool implementation in SecC
- Case studies: Casino, Regex matcher

Outlook

- Events with data
- Concurrency and lock invariants
- Context free, LTL, CTL?