Building a Symbolic Model Checker from Formal Language Description

Edmundo López Bóbeda and Didier Buchs

Friday, March 6th 2015 - Paris, France

UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES

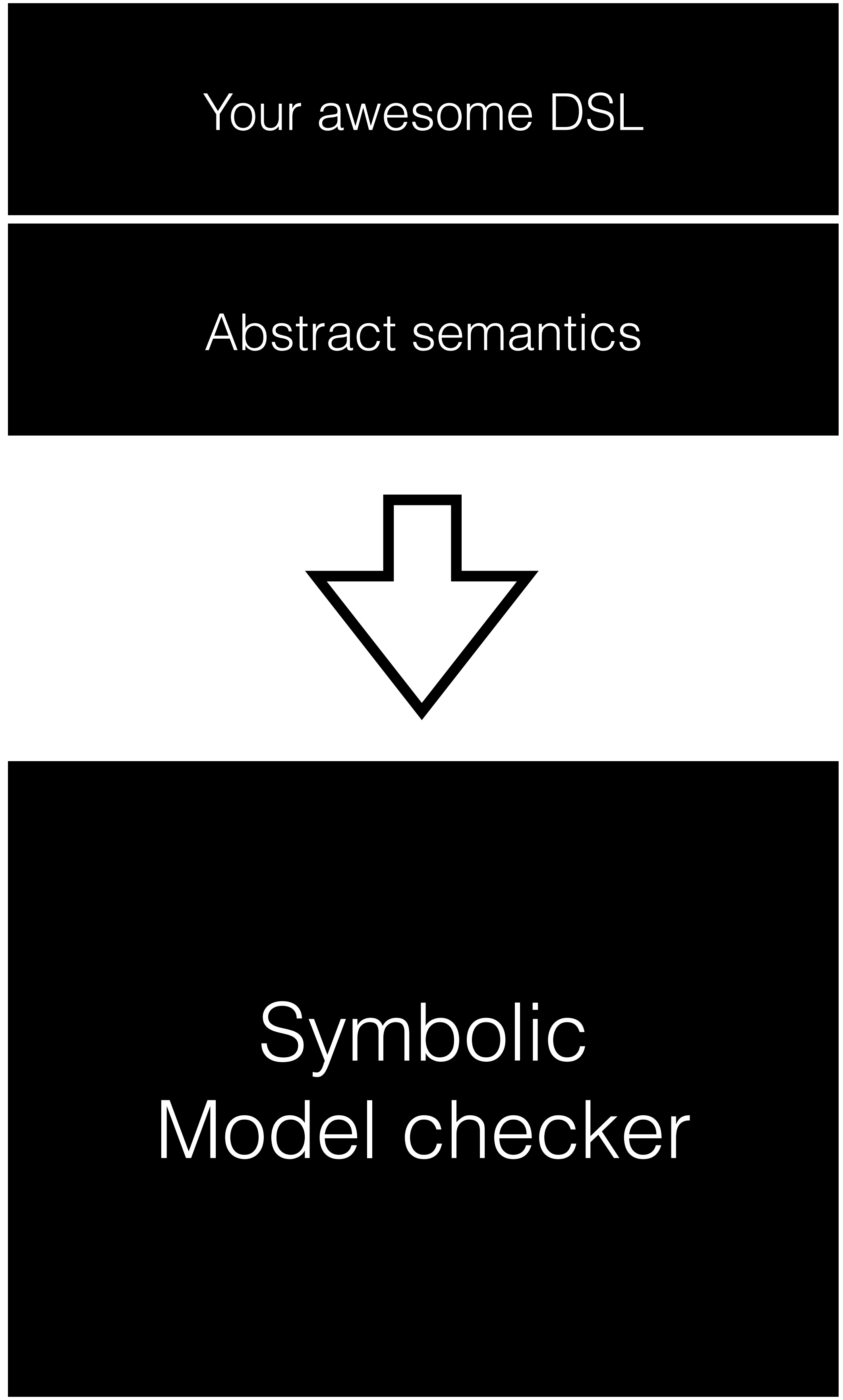
Méthodes Formelles pour les Systèmes Logiciels et Matériels

SMV

Your awesome DSL

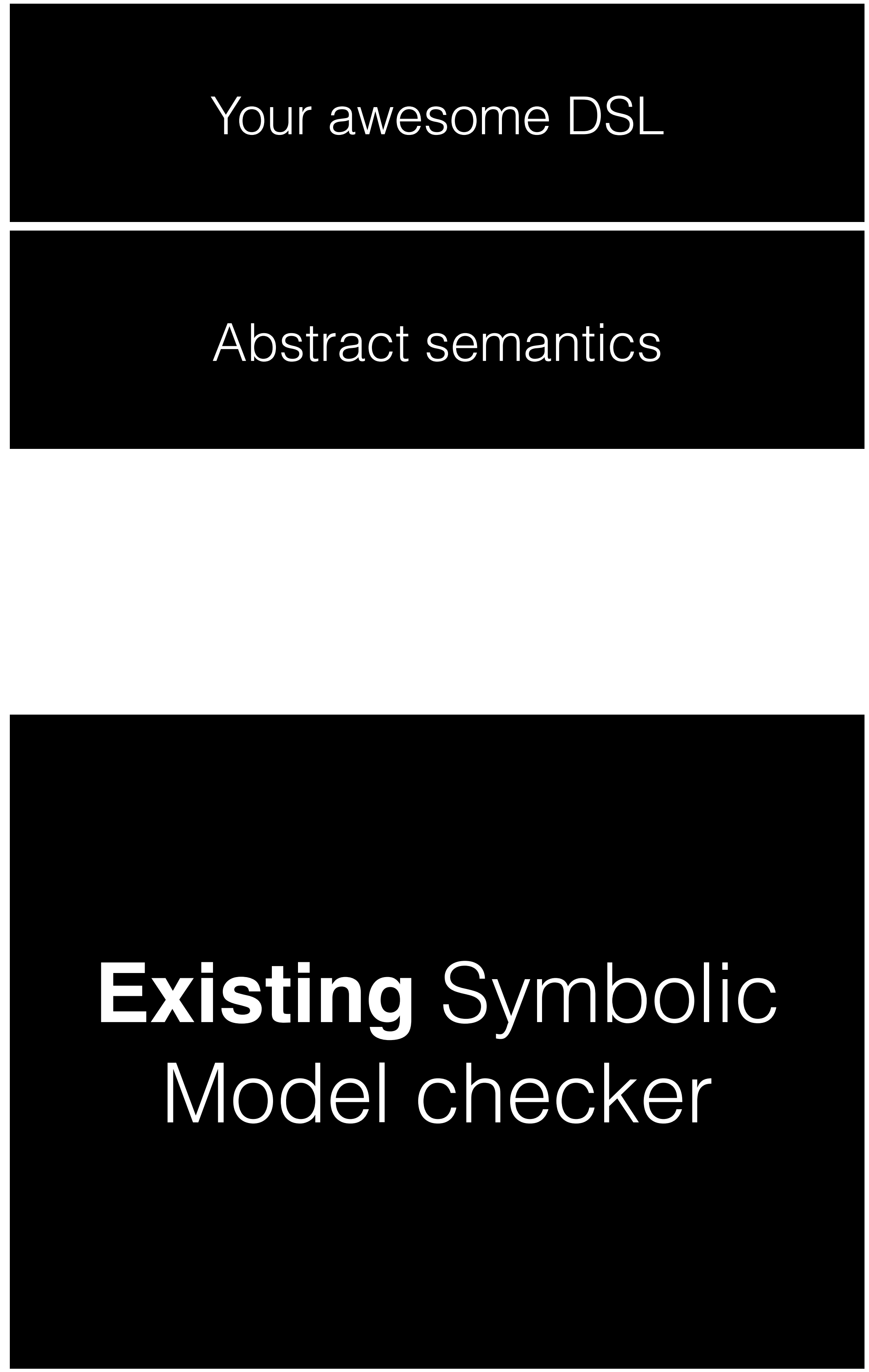
Your awesome DSL

Abstract semantics


Your awesome DSL
Abstract semantics
Symbolic
Model checker

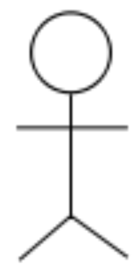
Your awesome DSL

Abstract semantics


Your awesome DSL
Abstract semantics
Existing Symbolic
Model checker

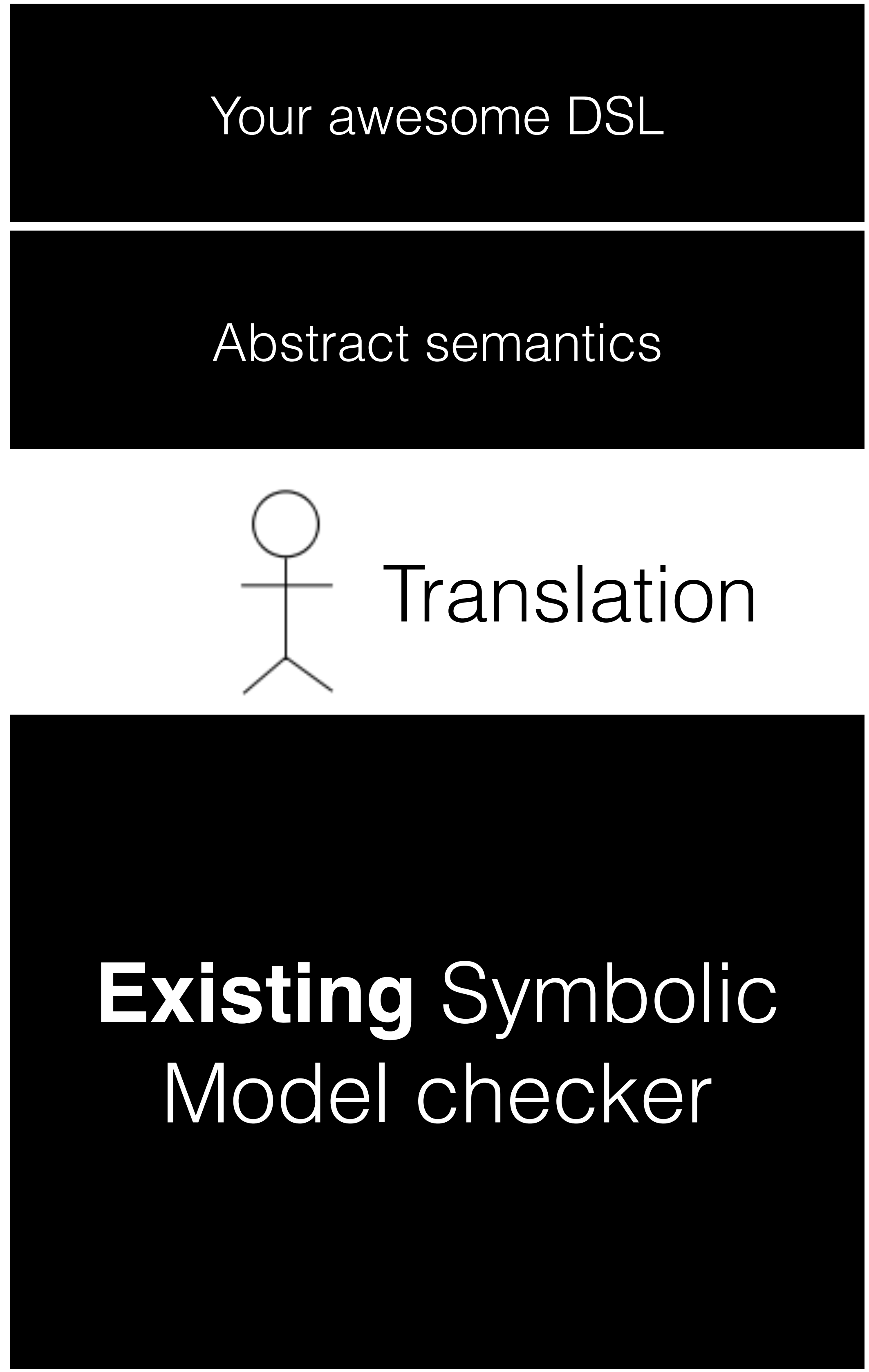
Your awesome DSL

Abstract semantics

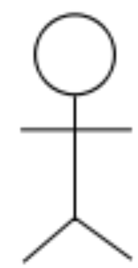


Translation

Existing Symbolic
Model checker

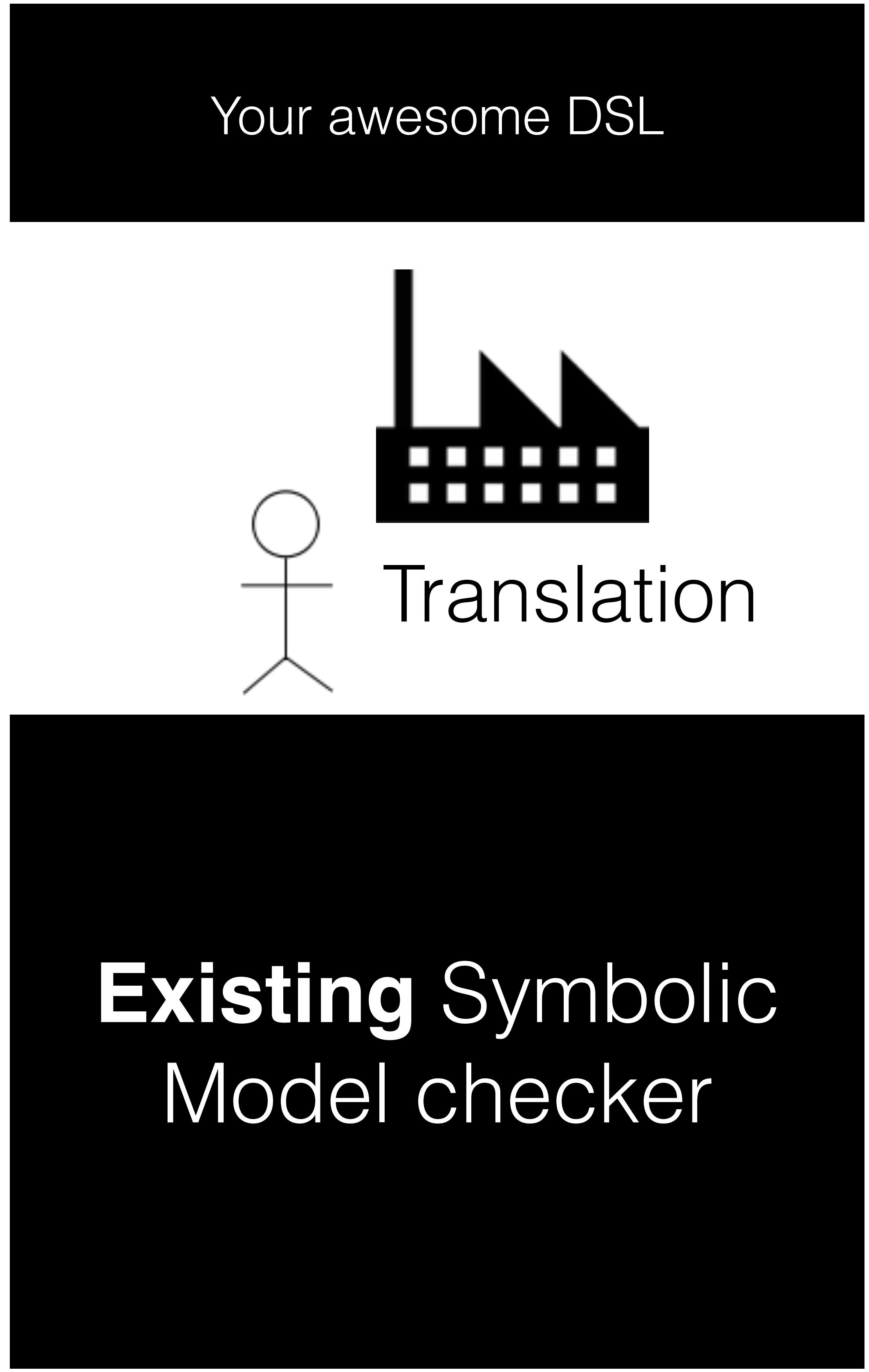

Your awesome DSL
Abstract semantics
Translation
Existing Symbolic
Model checker

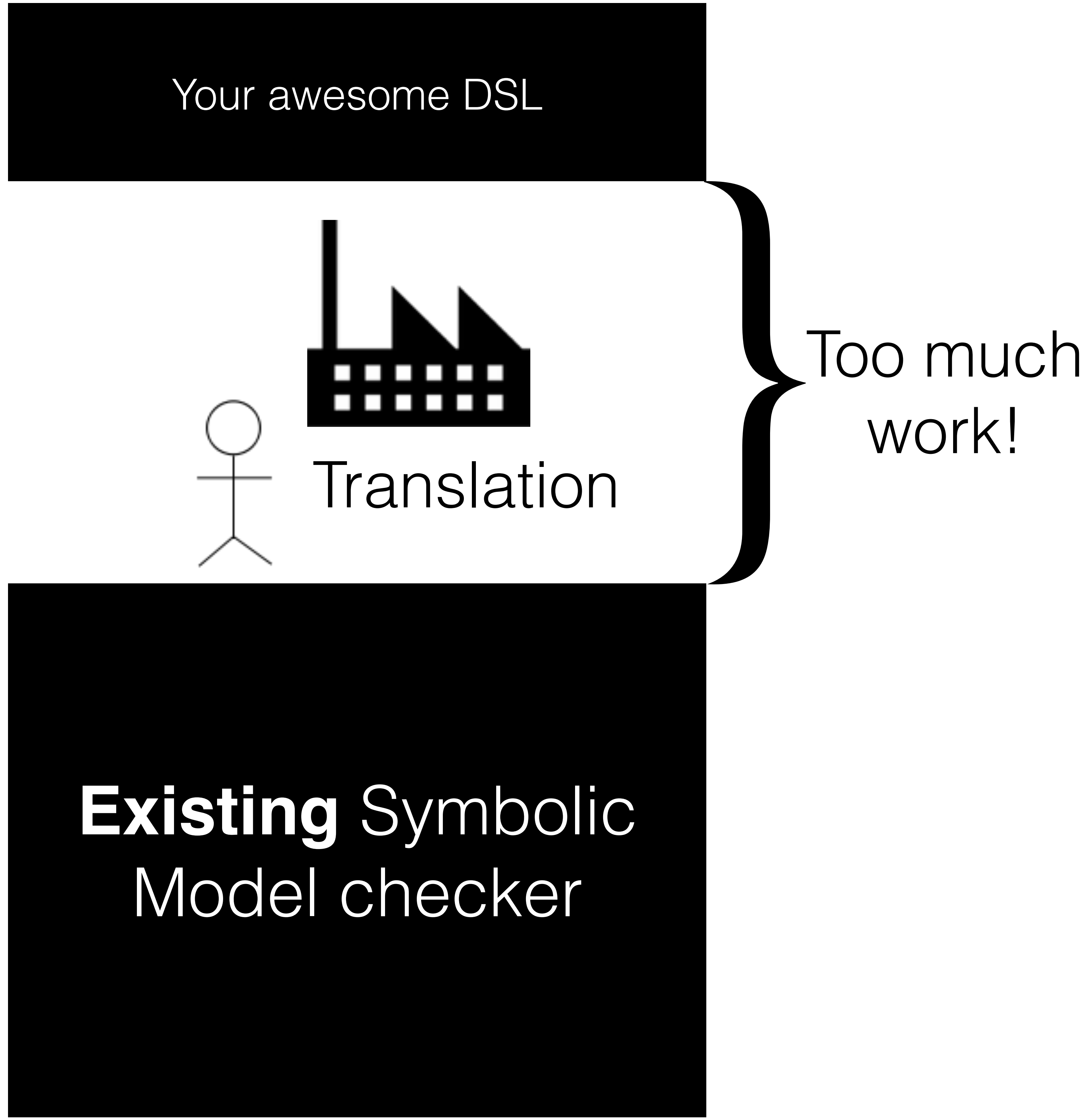
Your awesome DSL

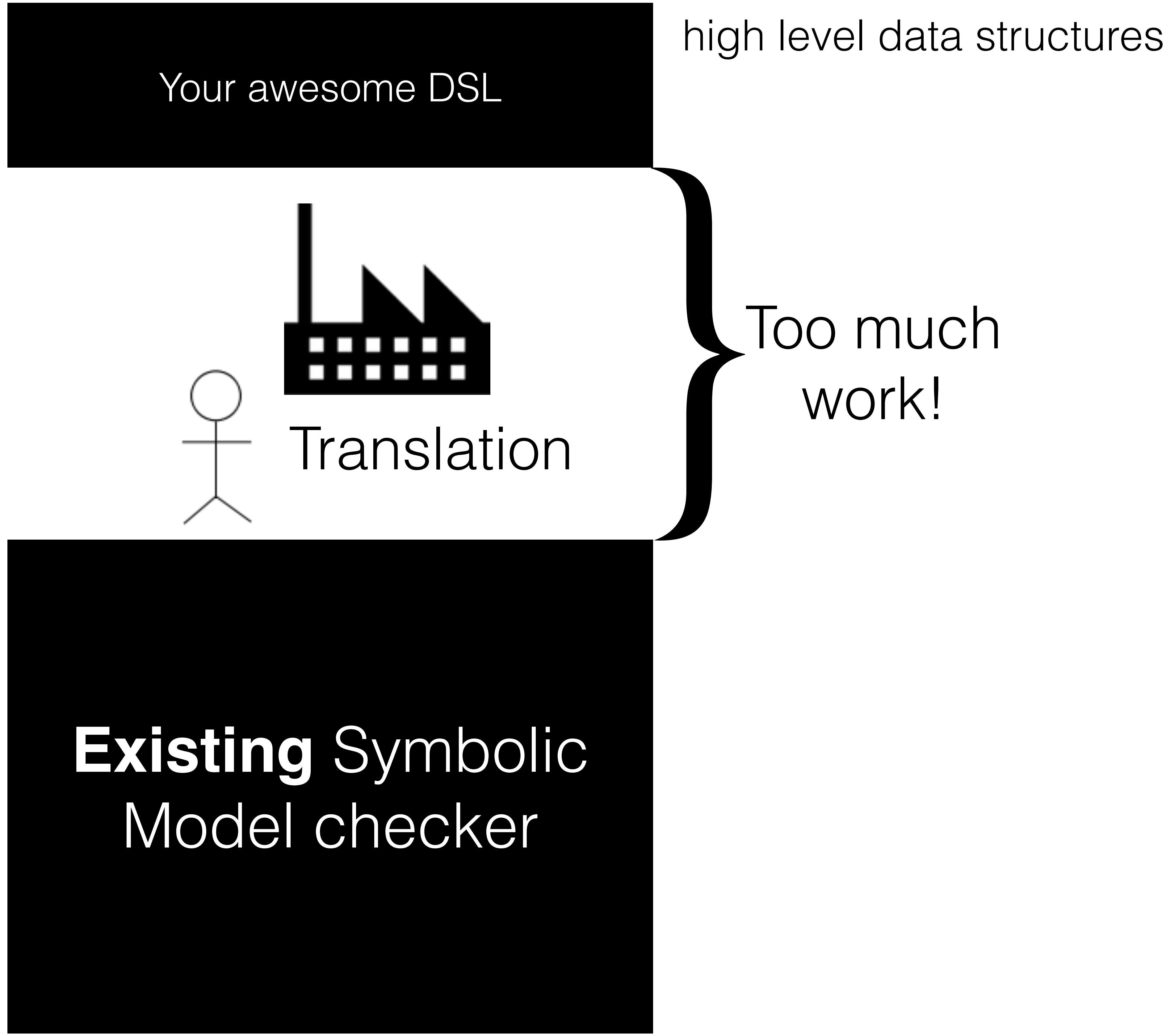


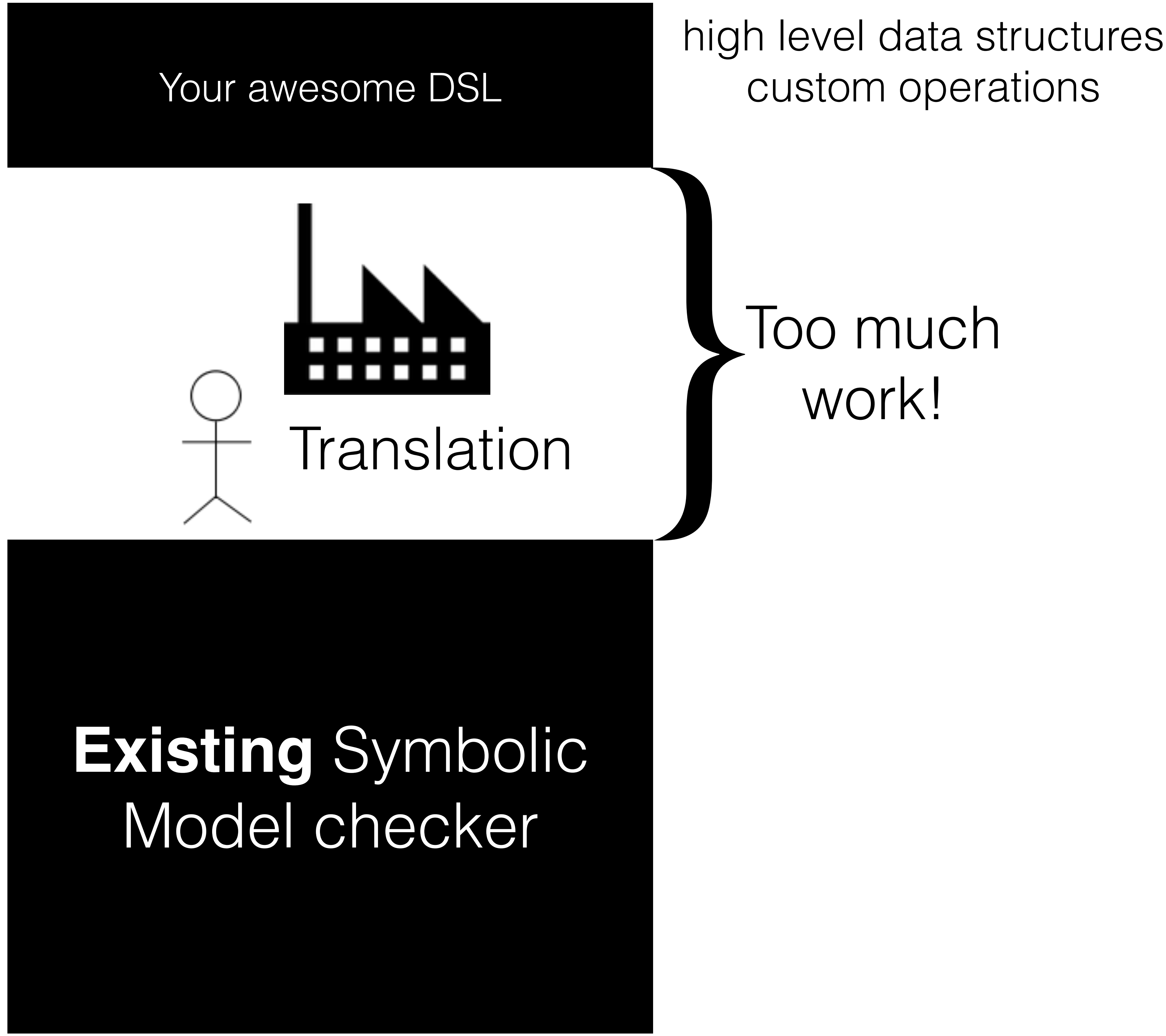
Translation

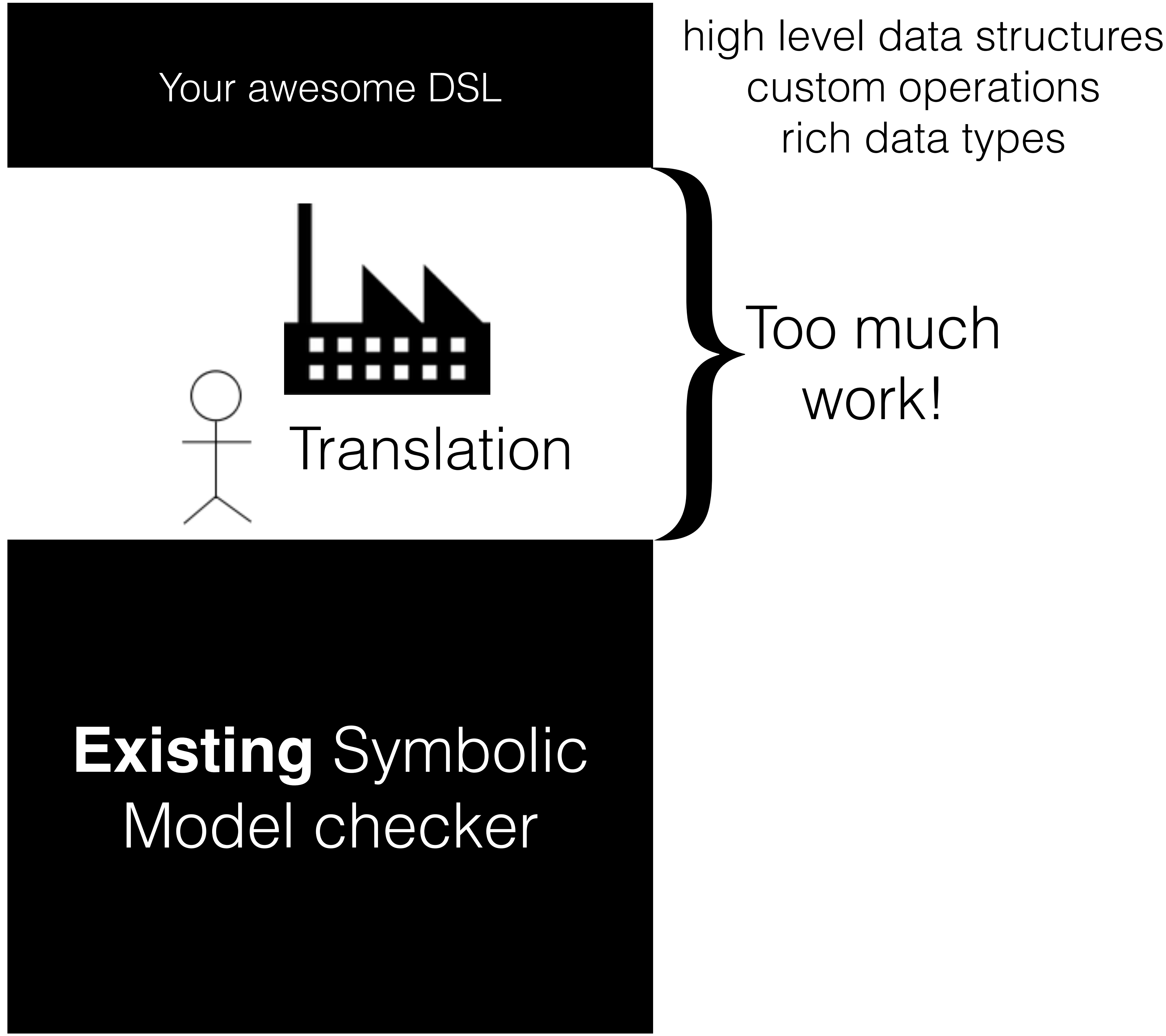
Existing Symbolic Model checker

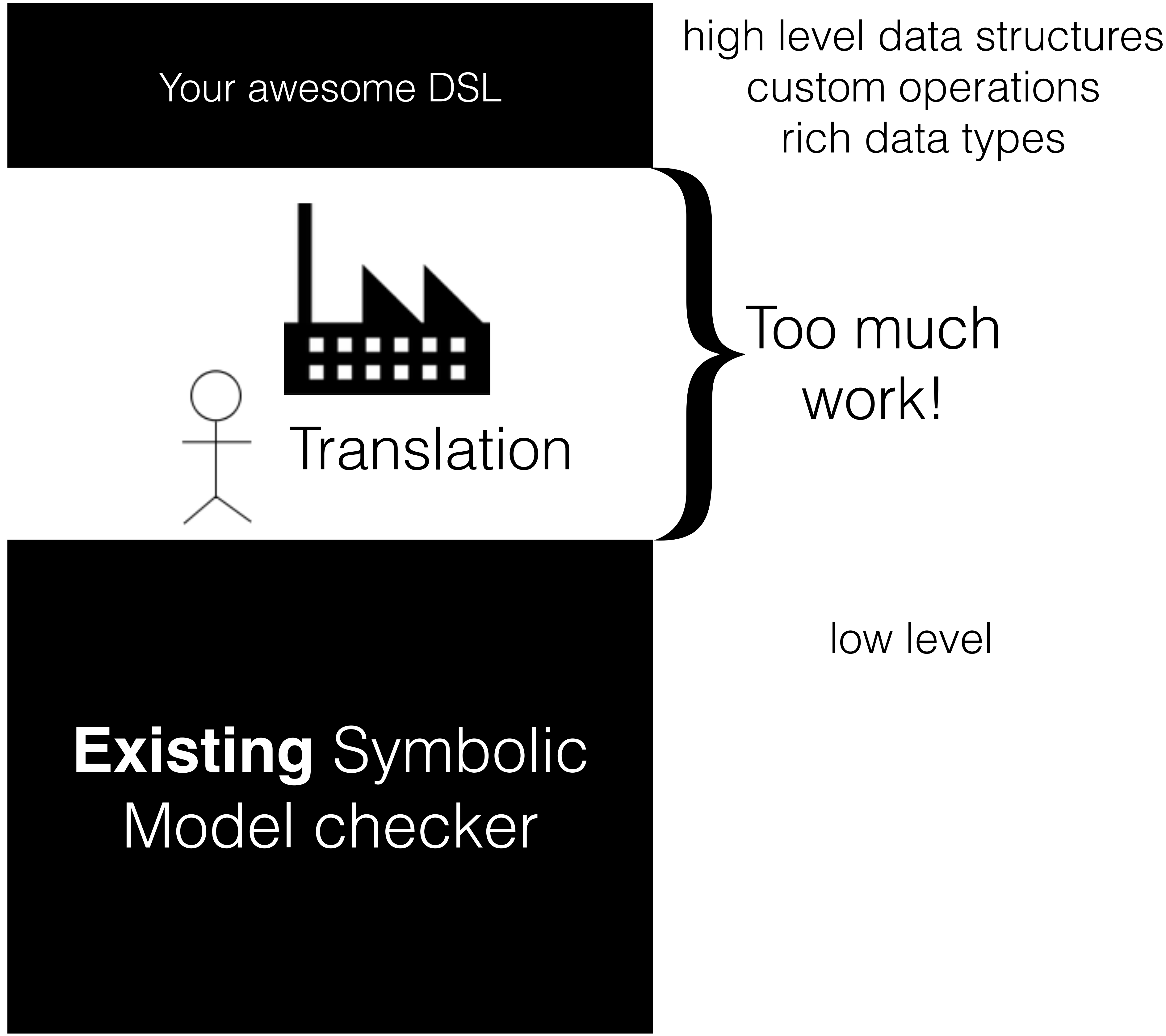

Your awesome DSL
Translation
Existing Symbolic
Model checker

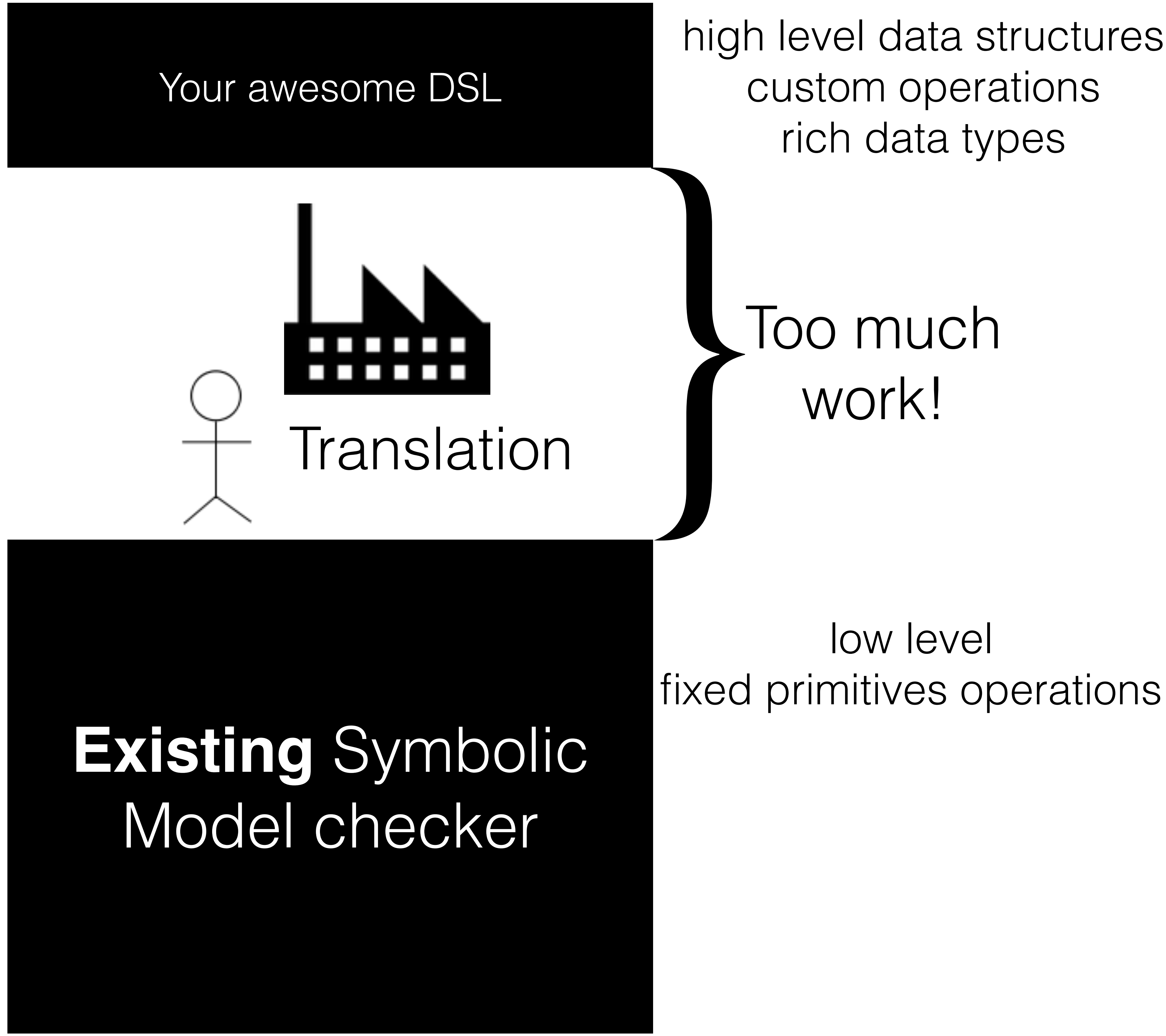

Your awesome DSL
Translation
Too much work!
Existing Symbolic Model checker

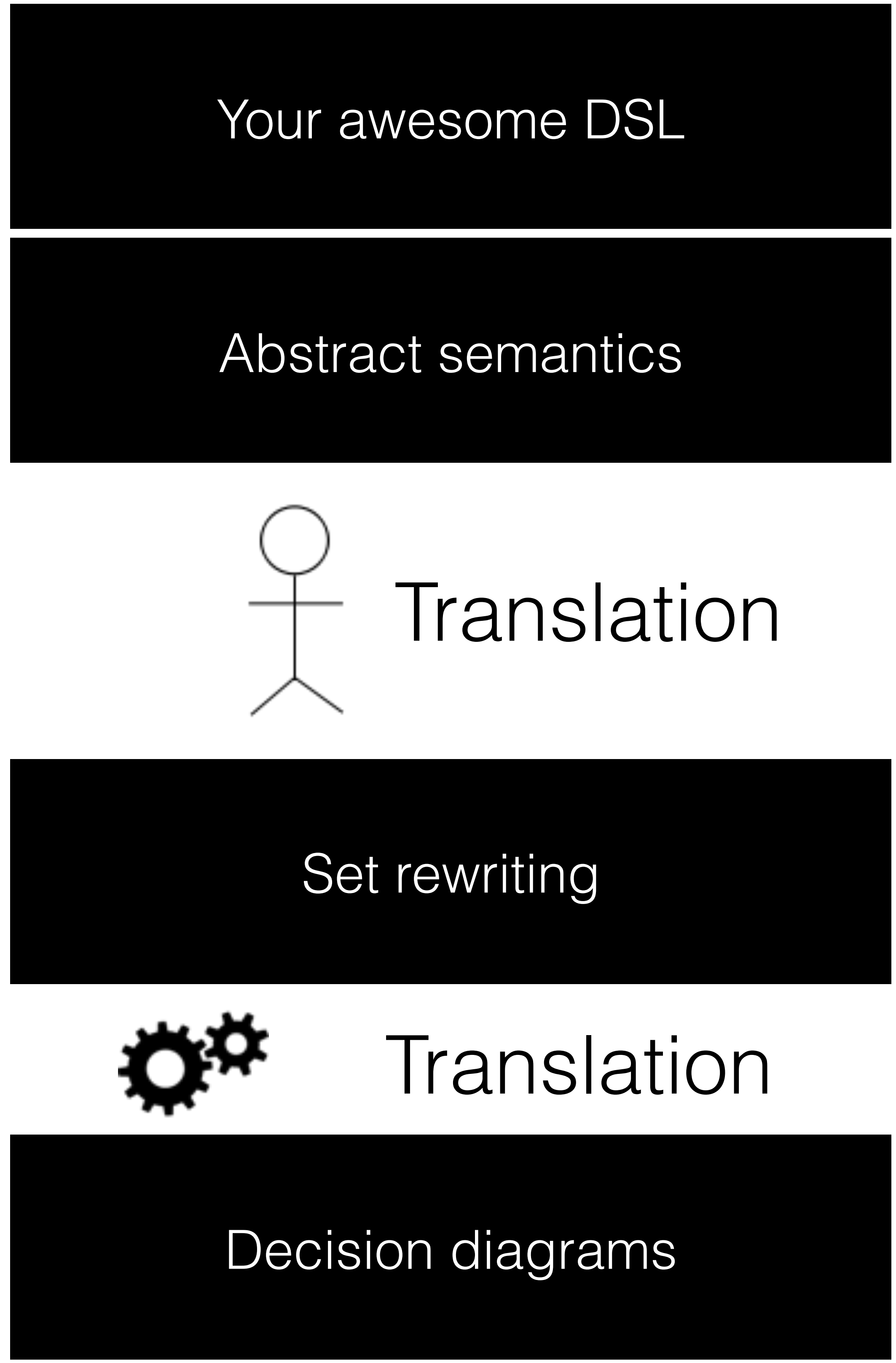

high level data structures
Your awesome DSL
Translation
Too much work!
Existing Symbolic Model checker

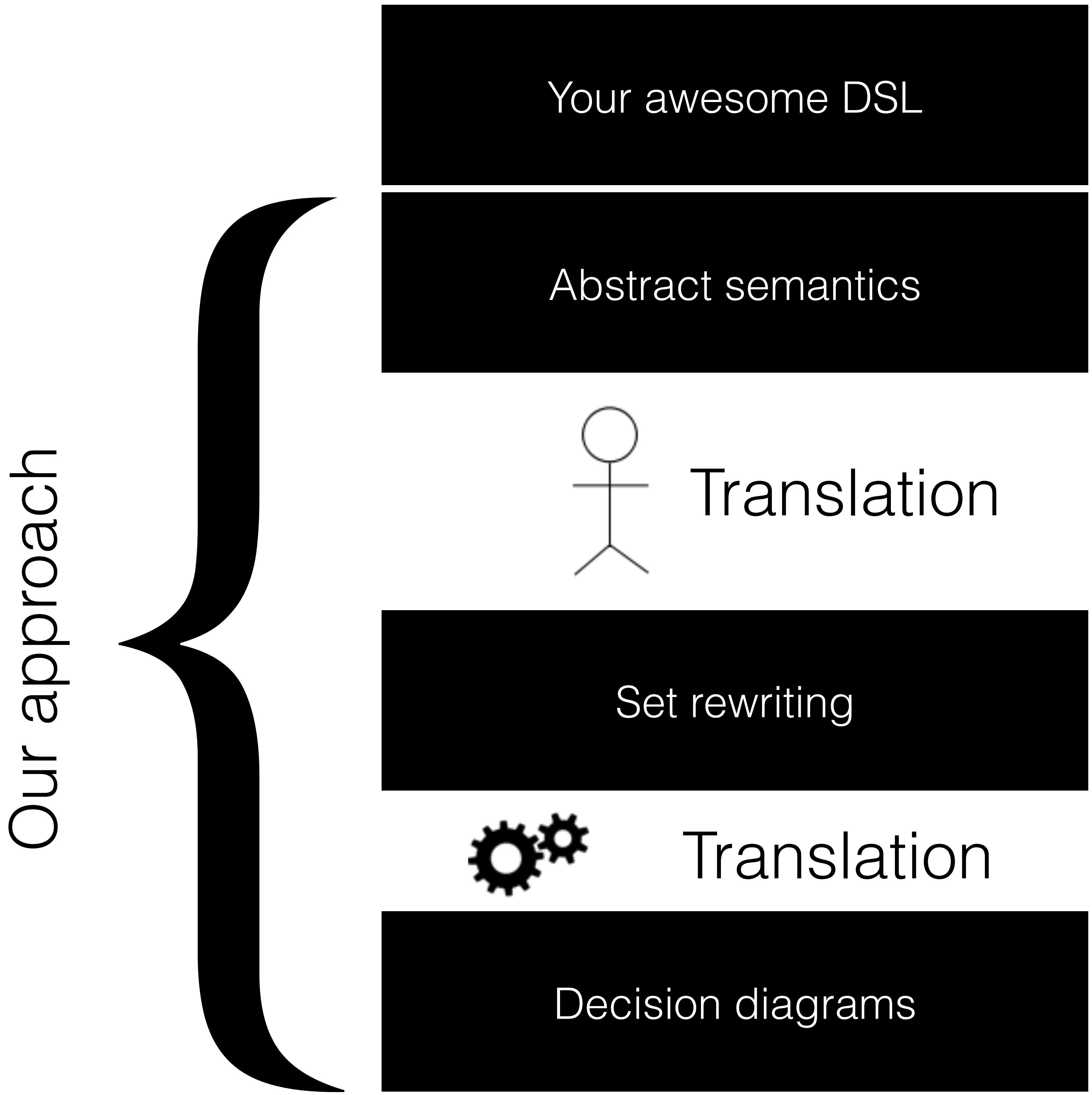

Your awesome DSL
high level data structures
custom operations
Translation
Too much
work!
Existing Symbolic
Model checker

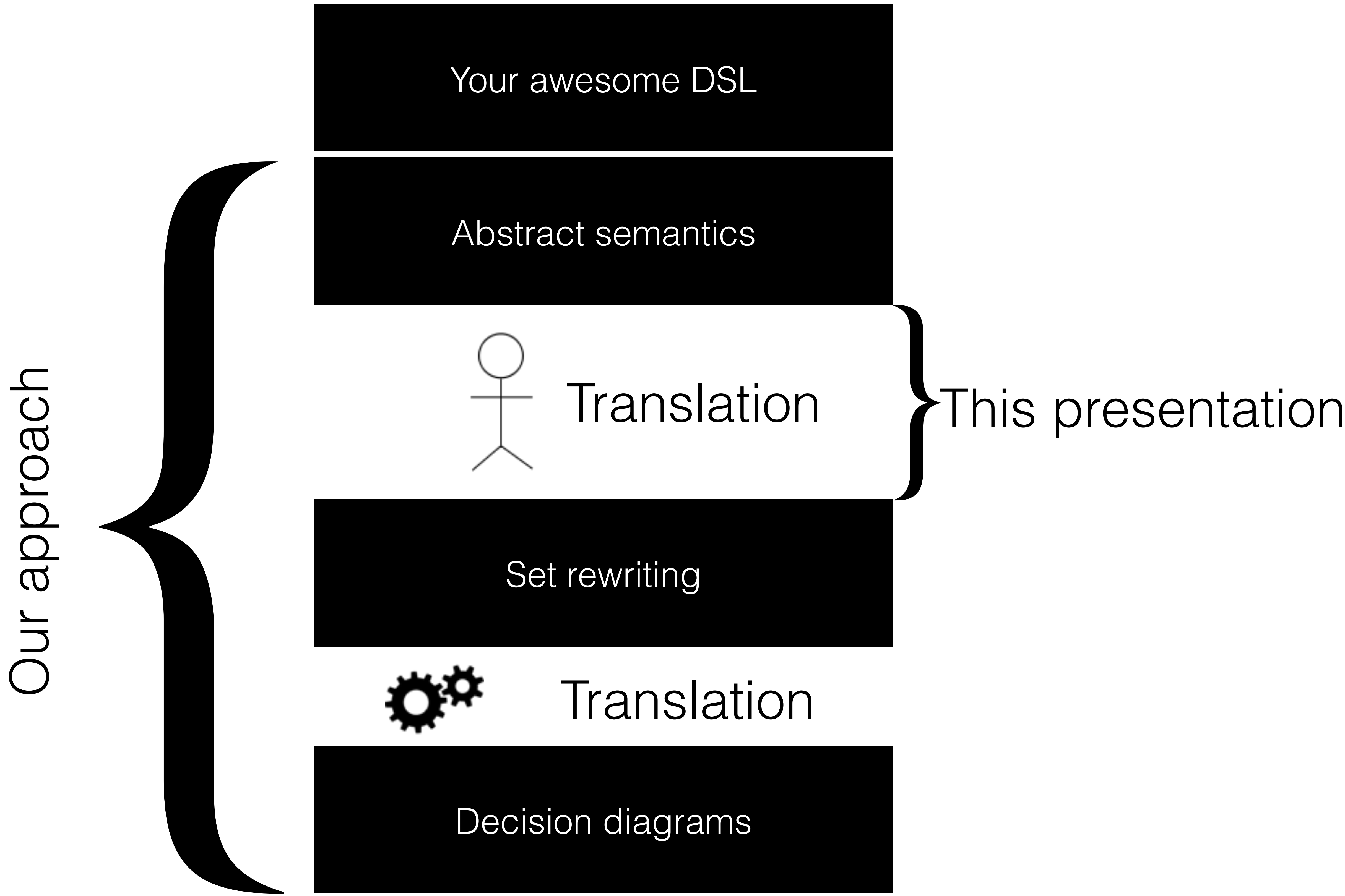

Your awesome DSL
high level data structures
custom operations
rich data types
Translation
Too much work!
Existing Symbolic Model checker


Your awesome DSL
high level data structures
custom operations
rich data types
Translation
Too much
work!
low level
Existing Symbolic
Model checker


Your awesome DSL
high level data structures
custom operations
rich data types
Translation
Too much
work!
Existing Symbolic
Model checker
low level
fixed primitives operations

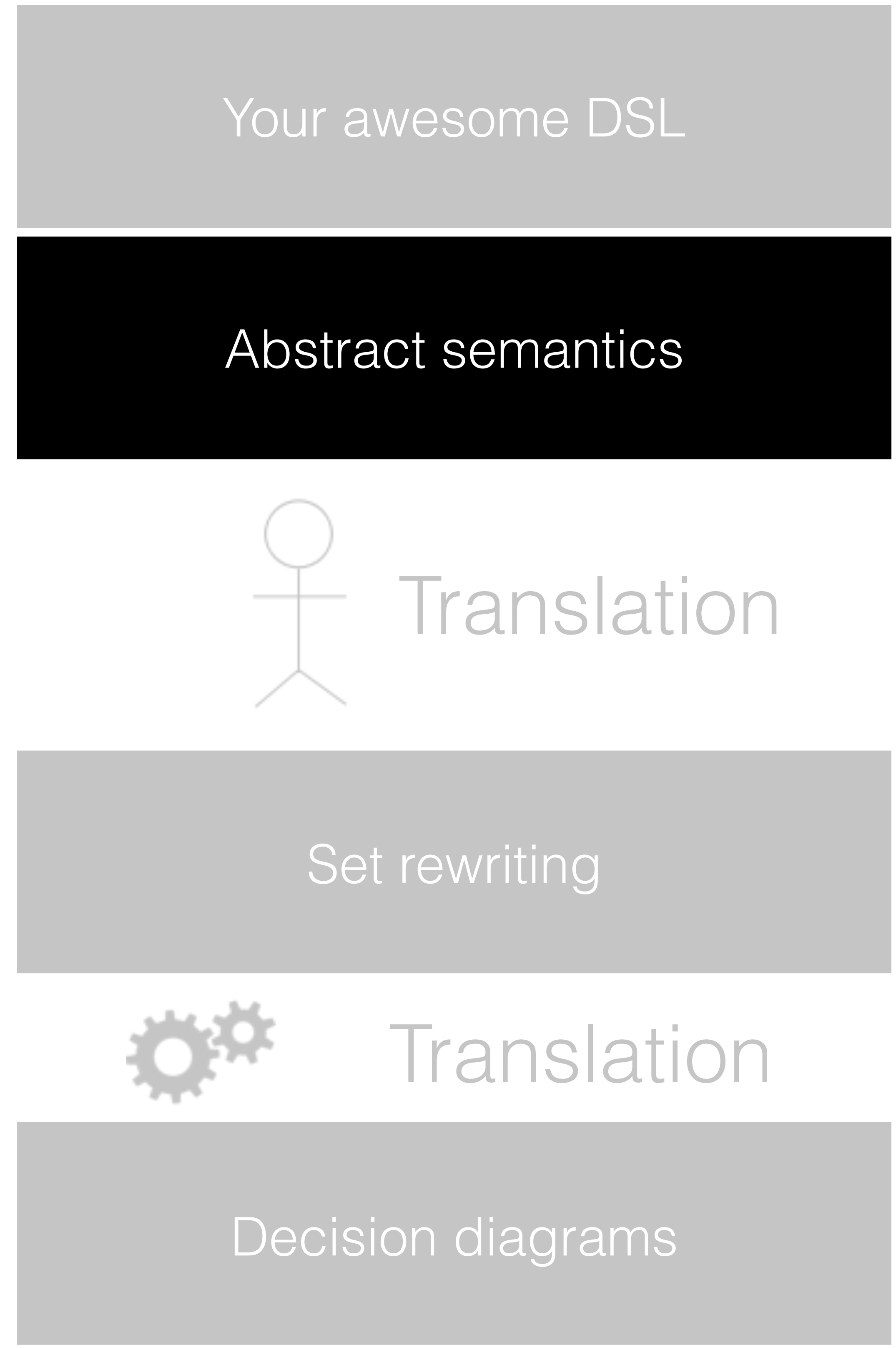

Your awesome DSL
Abstract semantics
Translation
Set rewriting
Translation
Decision diagrams


Your awesome DSL
Abstract semantics
Translation
Set rewriting
Translation
Decision diagrams
Our approach


Your awesome DSL
Abstract semantics
Translation
This presentation
Set rewriting
Translation
Decision diagrams
Our approach

Abstract semantics

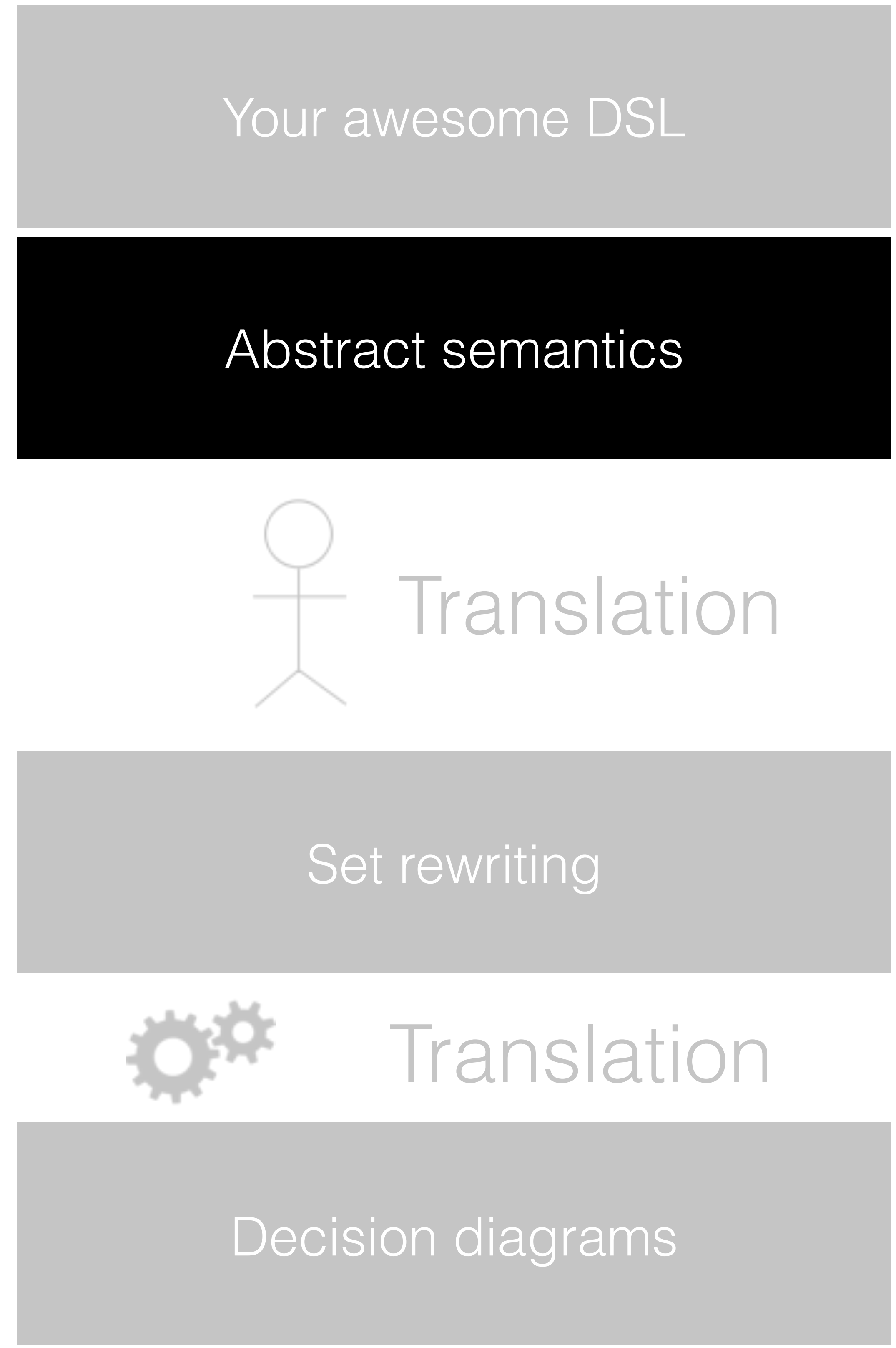
In context



Abstract semantics

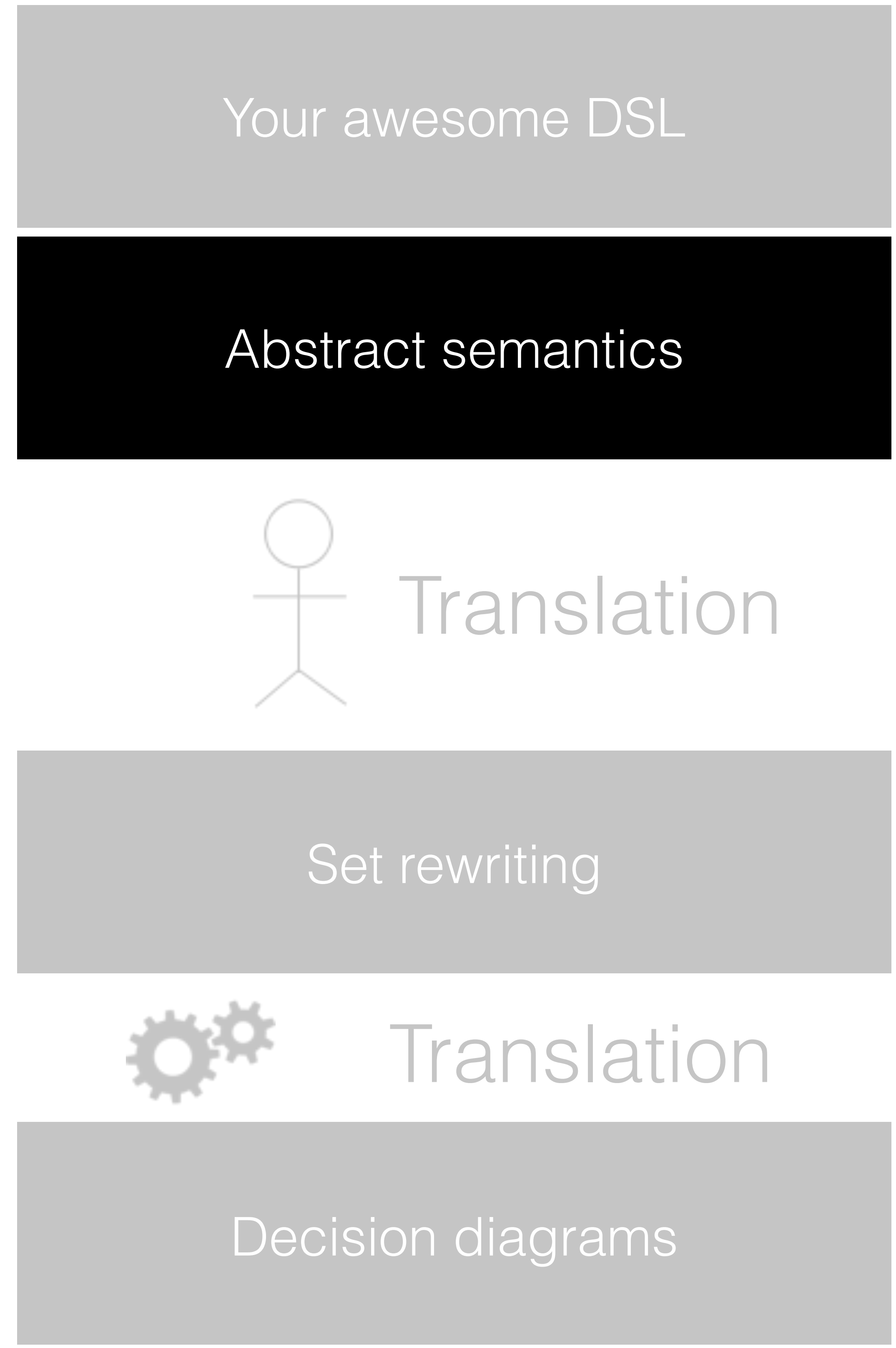
In context

- High level representation



Abstract semantics

In context



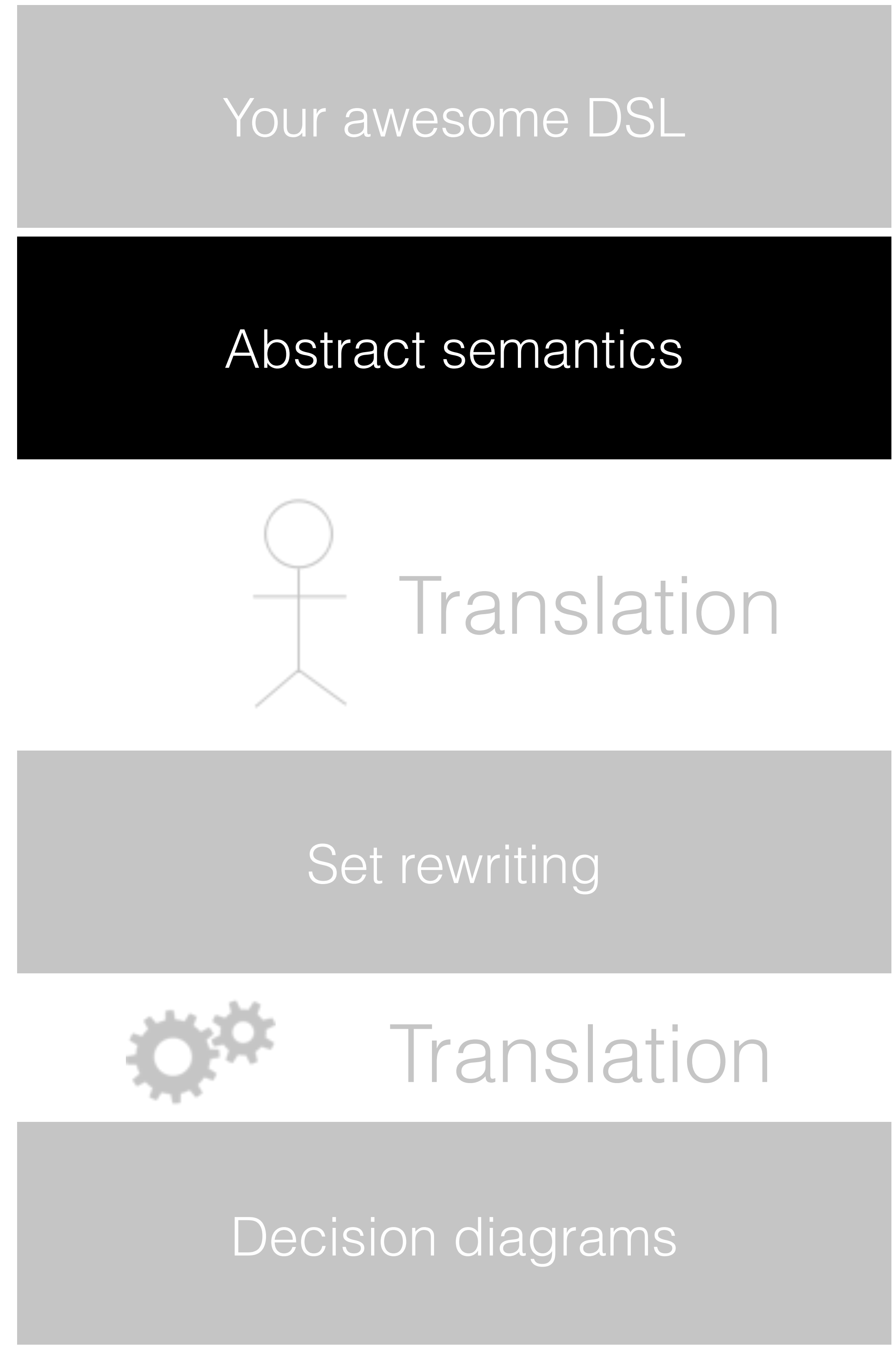
- High level representation
- Suitable for humans

Abstract semantics

In context

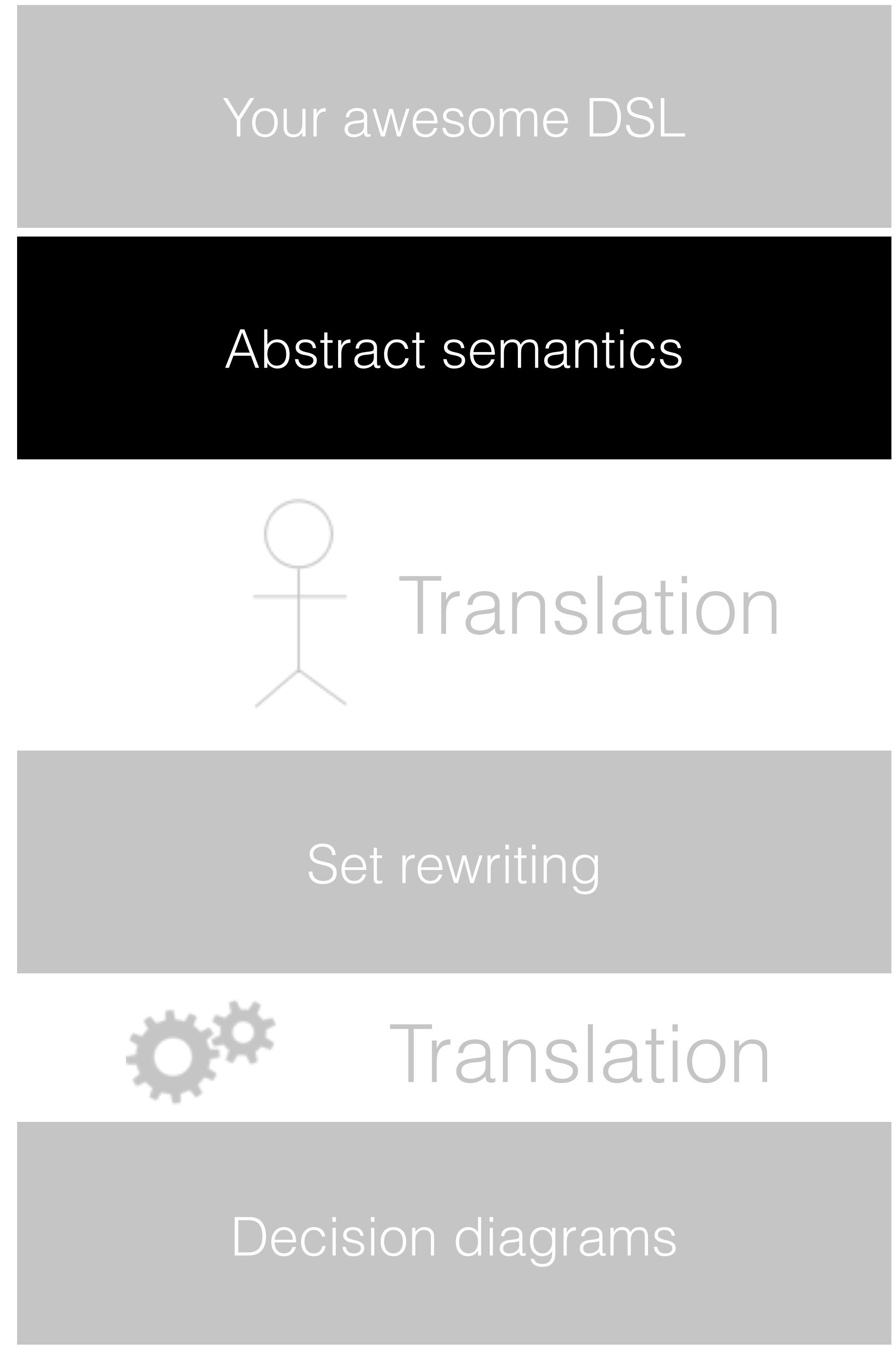
- We use Simple SOS rules:

$$\frac{c(s)}{s \to a(s)}$$



Abstract semantics

In context



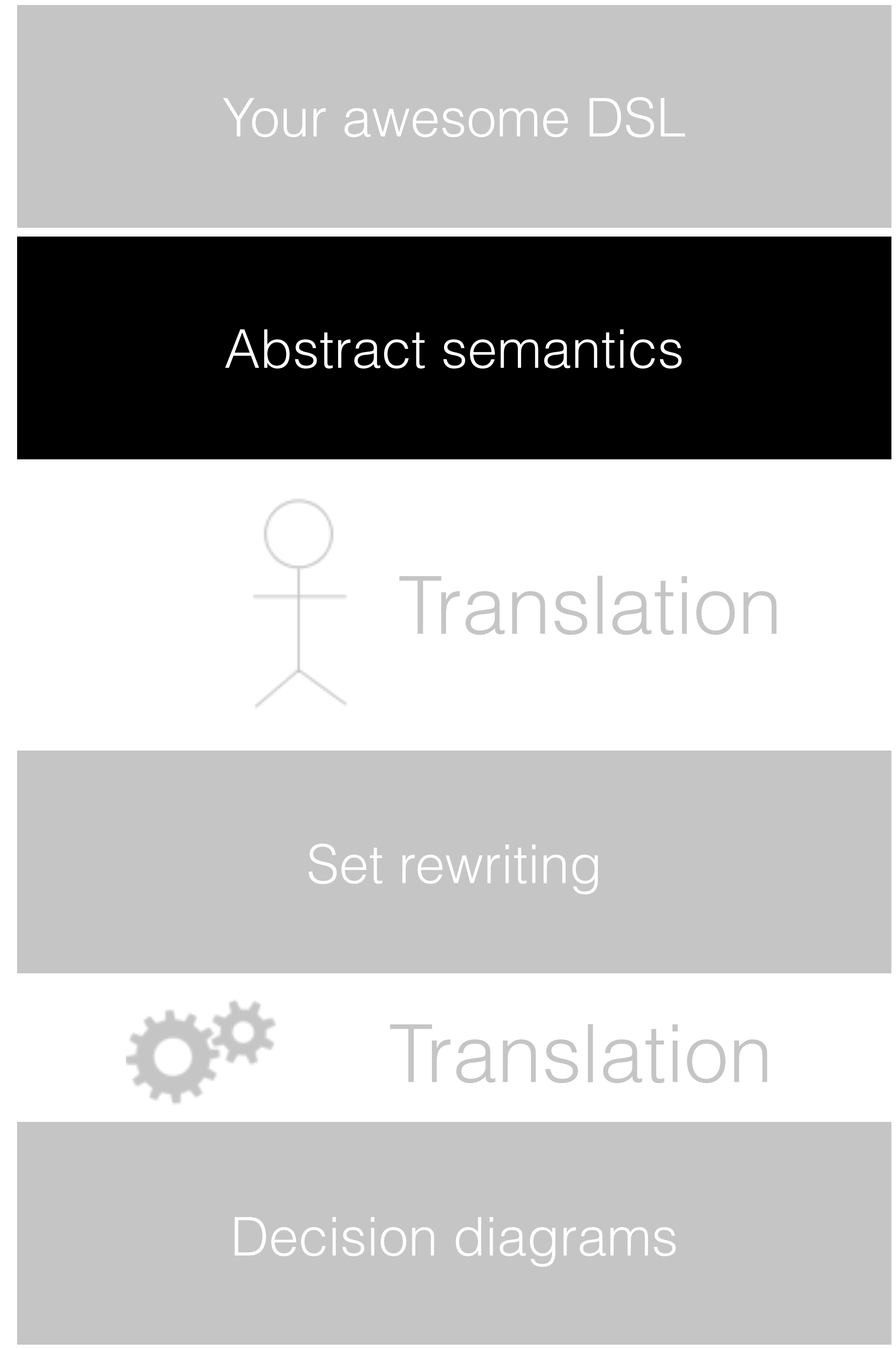
- We use Simple SOS rules:

$$\frac{c(s)}{s \rightarrow a(s)}$$

- s ∈ States

Abstract semantics

In context



- We use Simple SOS rules:

$$\frac{c(s)}{s \rightarrow a(s)}$$

- s ∈ States
- c: States ⟶ 𝔹, c ∈ Cond

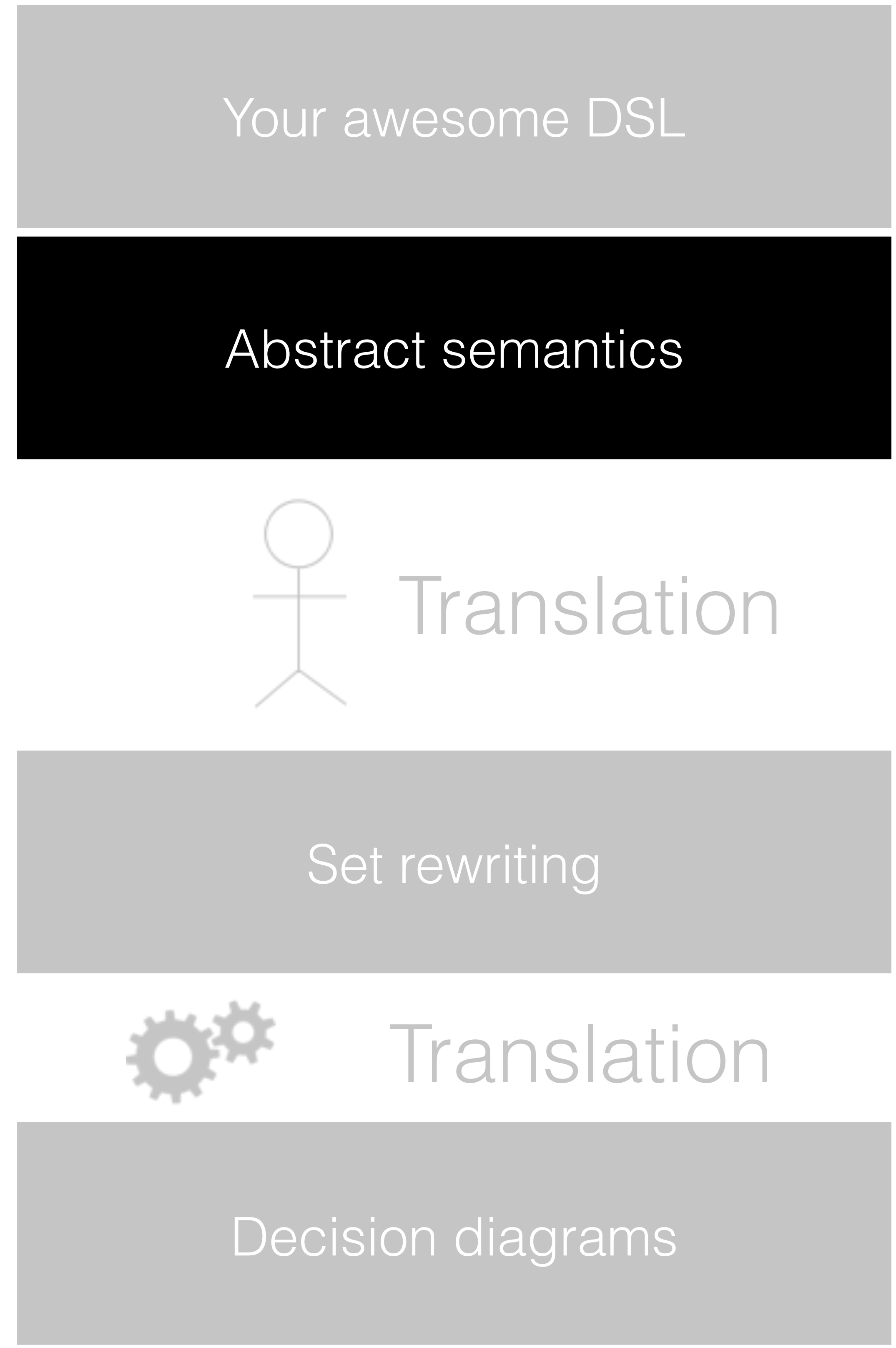
Abstract semantics

In context

- We use Simple SOS rules:

$$\frac{c(s)}{s \rightarrow a(s)}$$

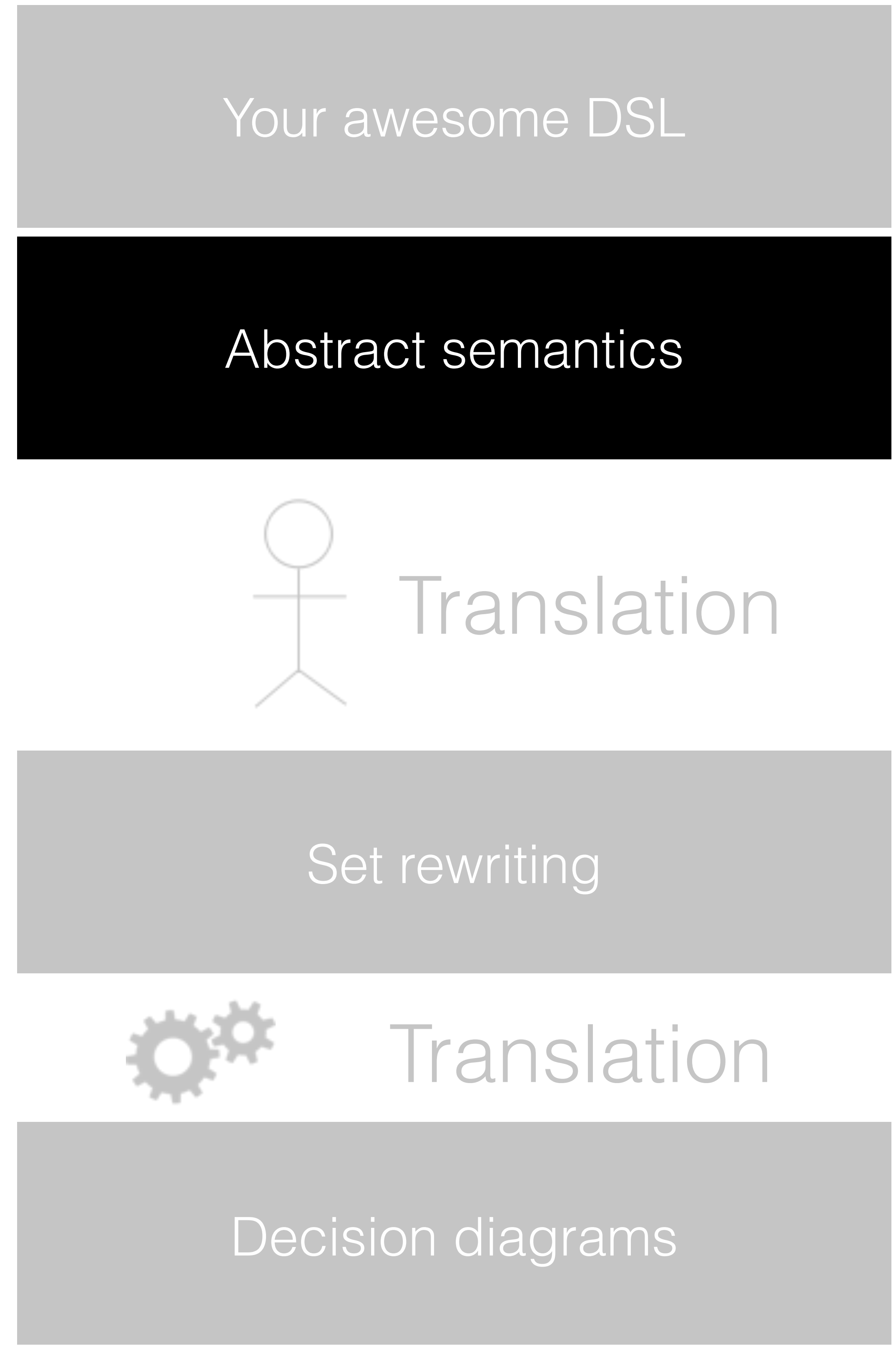
- s ∈ States
- c: States ⟶ $\mathbb{B}$, c ∈ Cond
- a: States ⟶ States, a ∈ Act



Abstract semantics

In context

$$\frac{c(s)}{s \rightarrow a(s)}$$

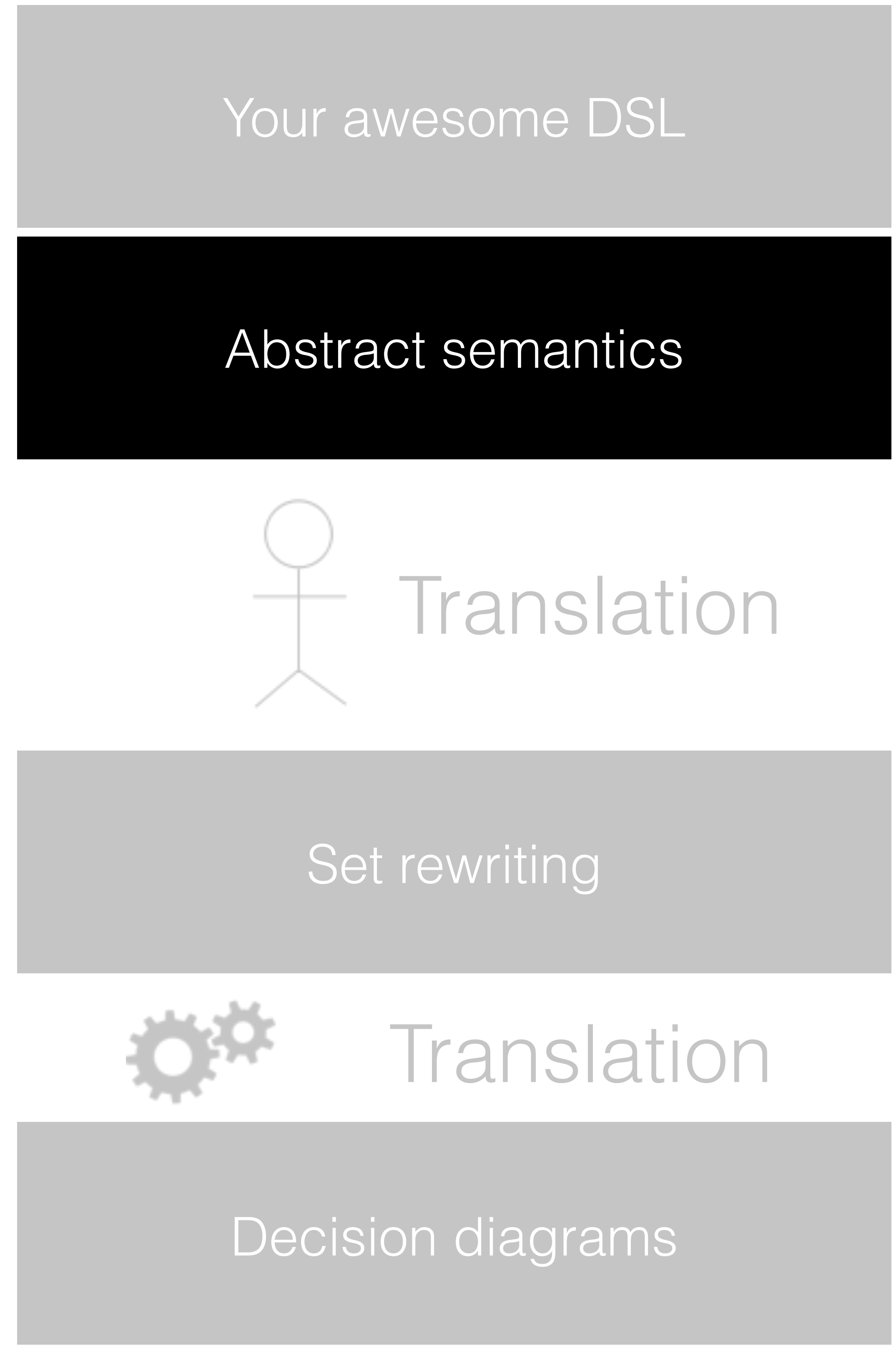


Abstract semantics

In context

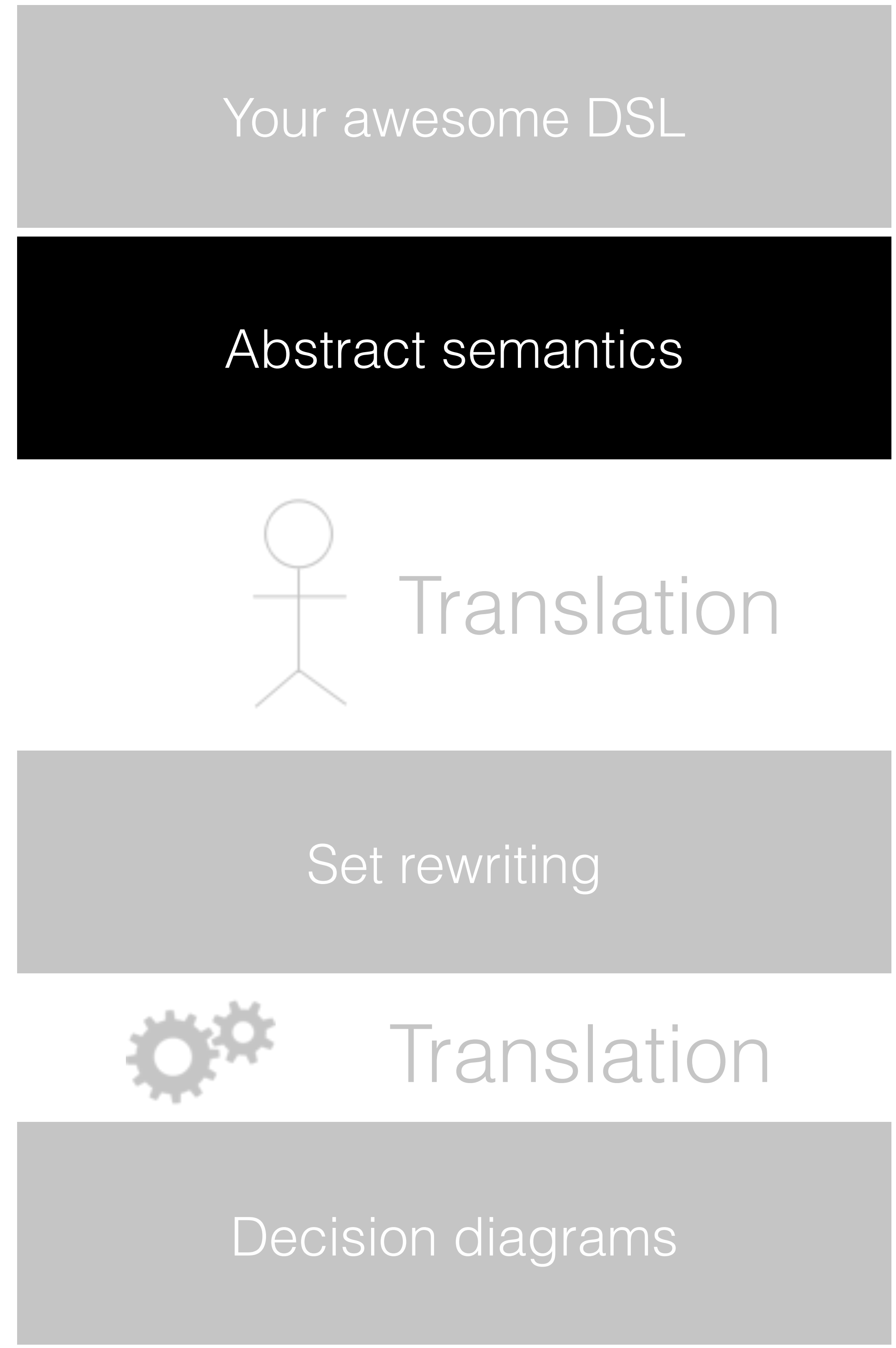
$$\frac{c(s)}{s \to a(s)}$$

- $c_1 \wedge c_2 \in \mathrm{Cond}$



Abstract semantics

In context



$$\frac{c(s)}{s \to a(s)}$$

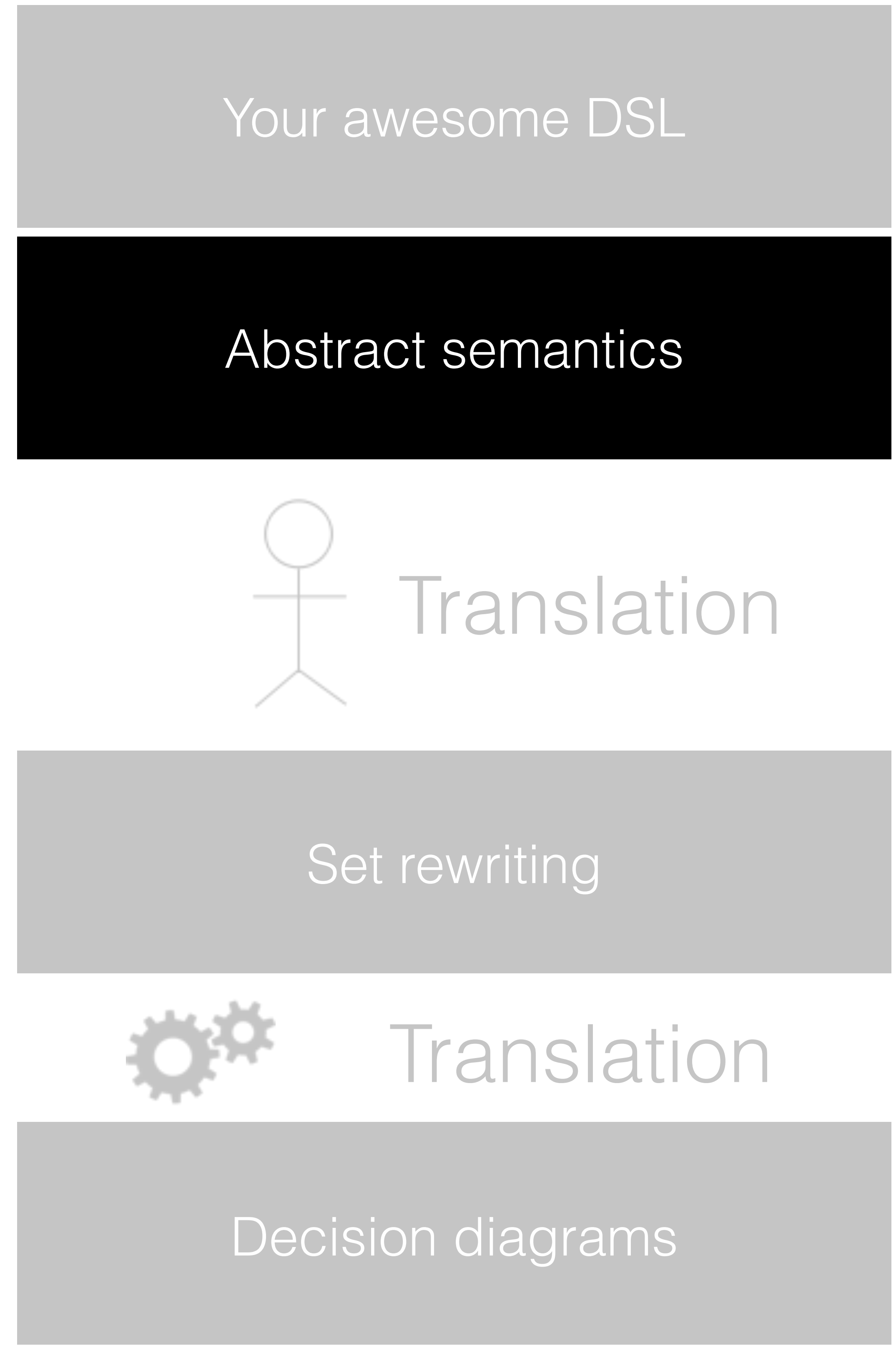
- $c_1 \wedge c_2 \in \text{Cond}$
- $c \circ a \in \text{Cond}$

Abstract semantics

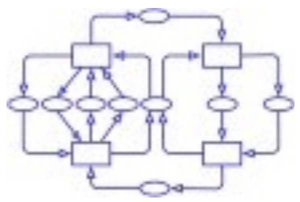
In context

$$\frac{c(s)}{s \to a(s)}$$

- $c_1 \wedge c_2 \in$ Cond
- $c \circ a \in$ Cond
- $a \circ a \in$ Act

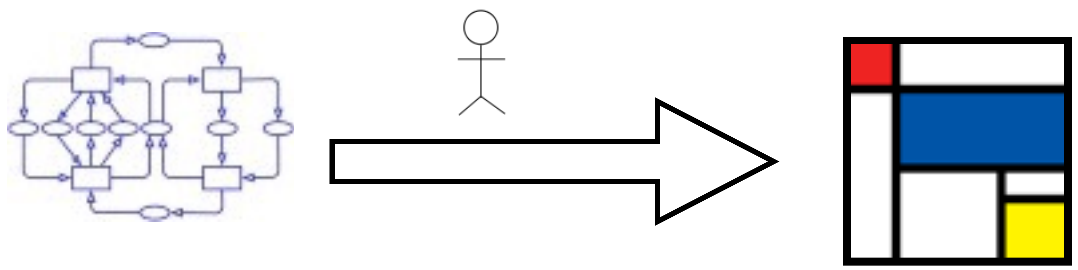


Goal

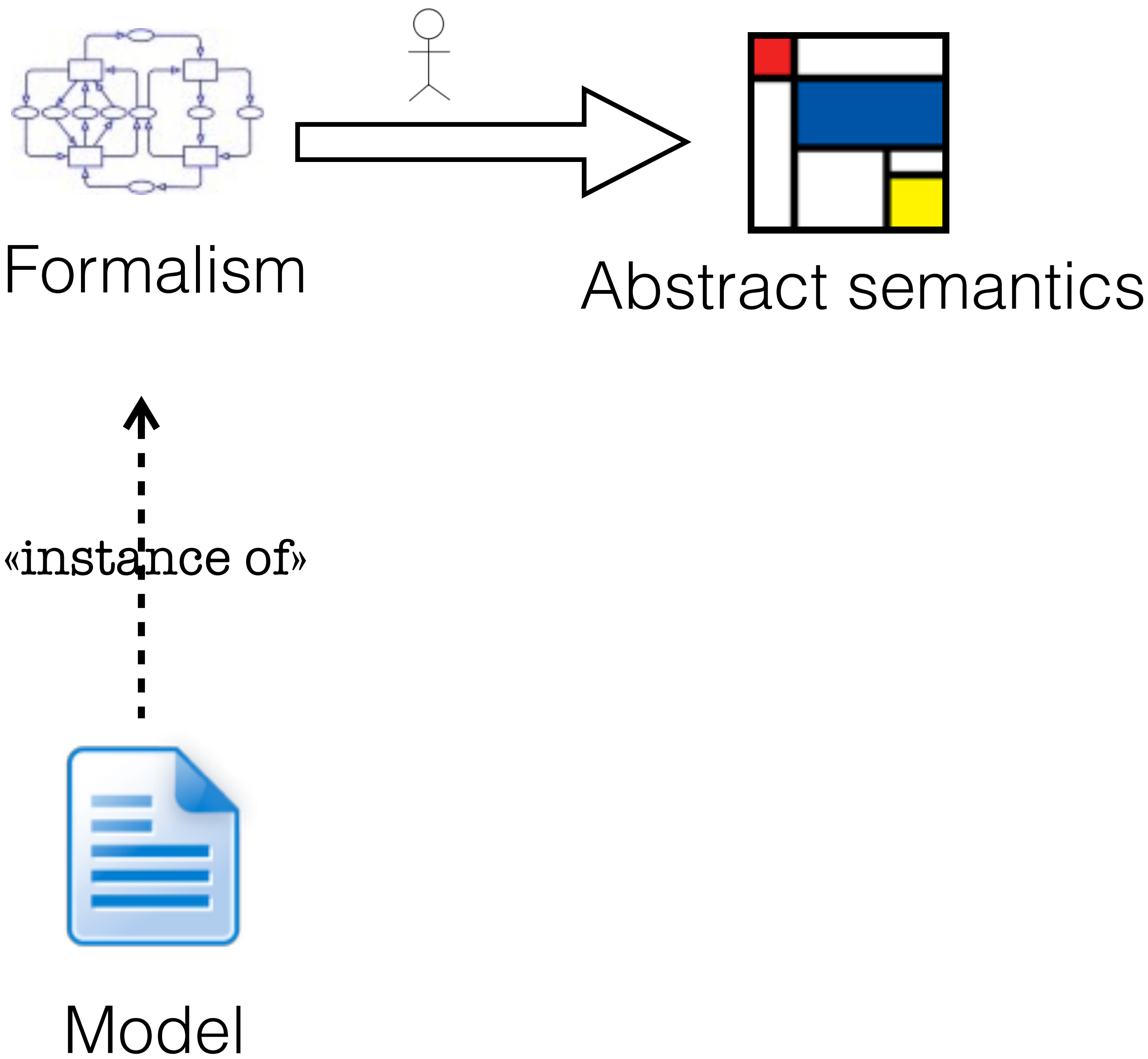


Formalism

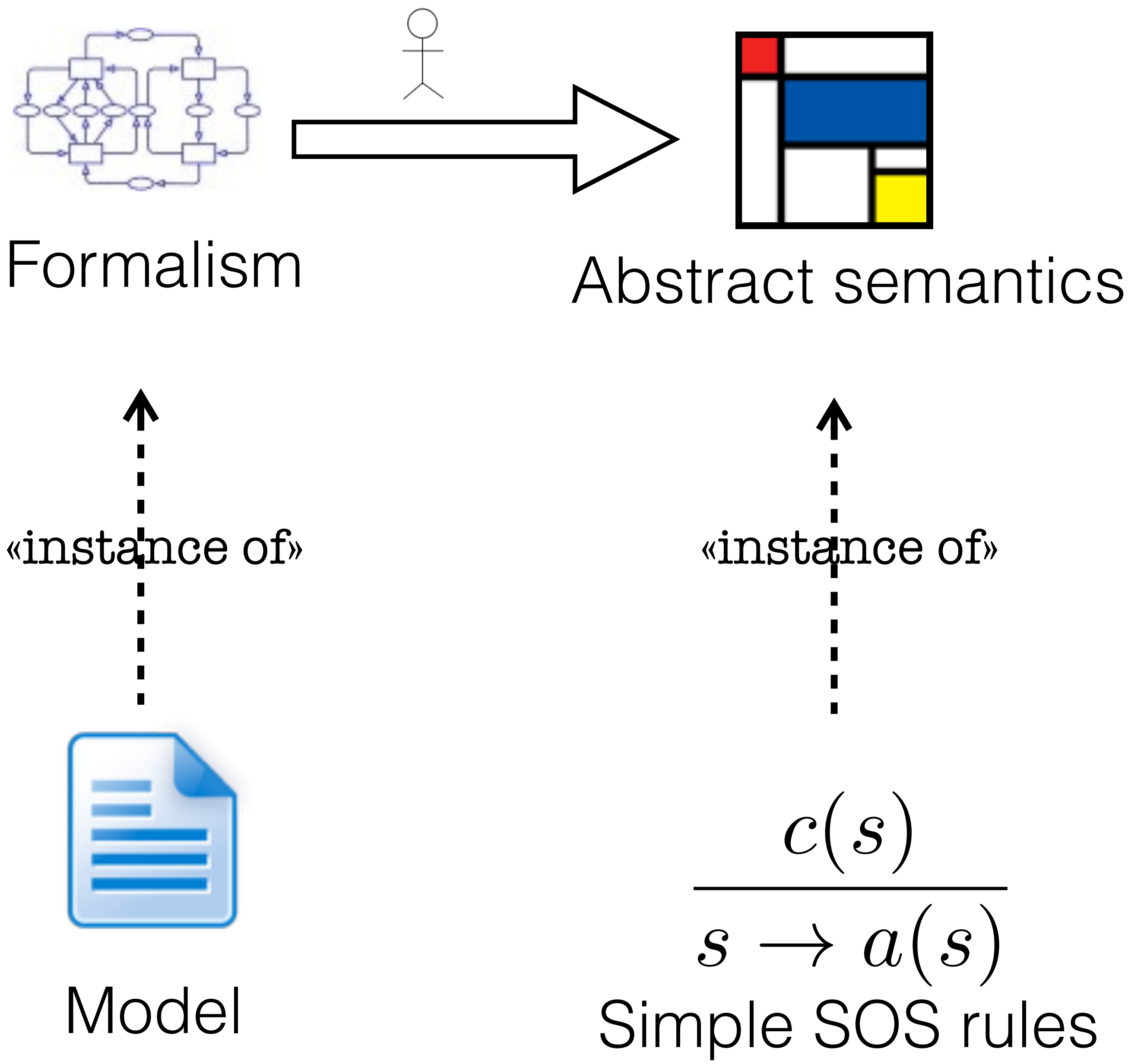
Goal



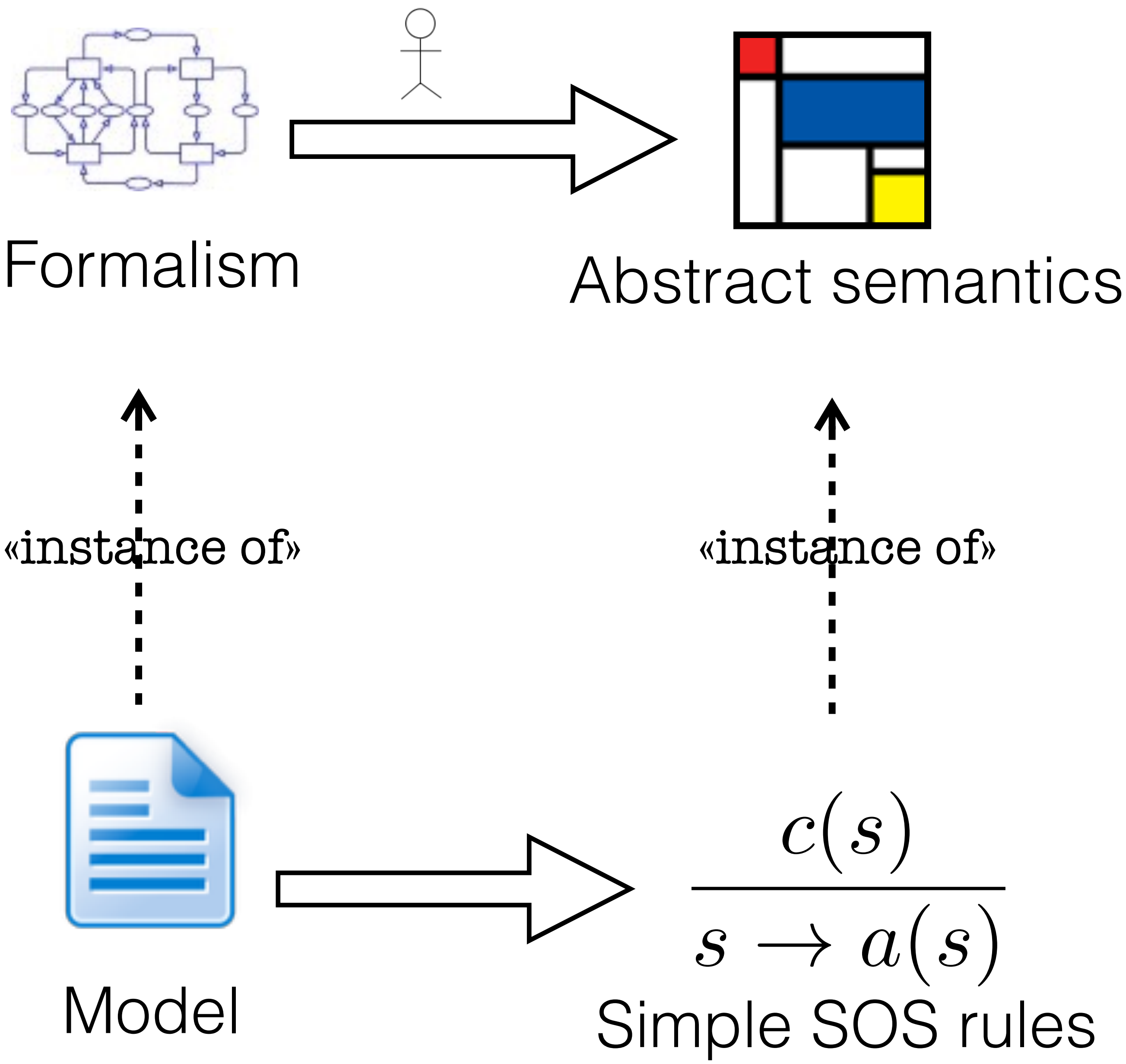
Goal



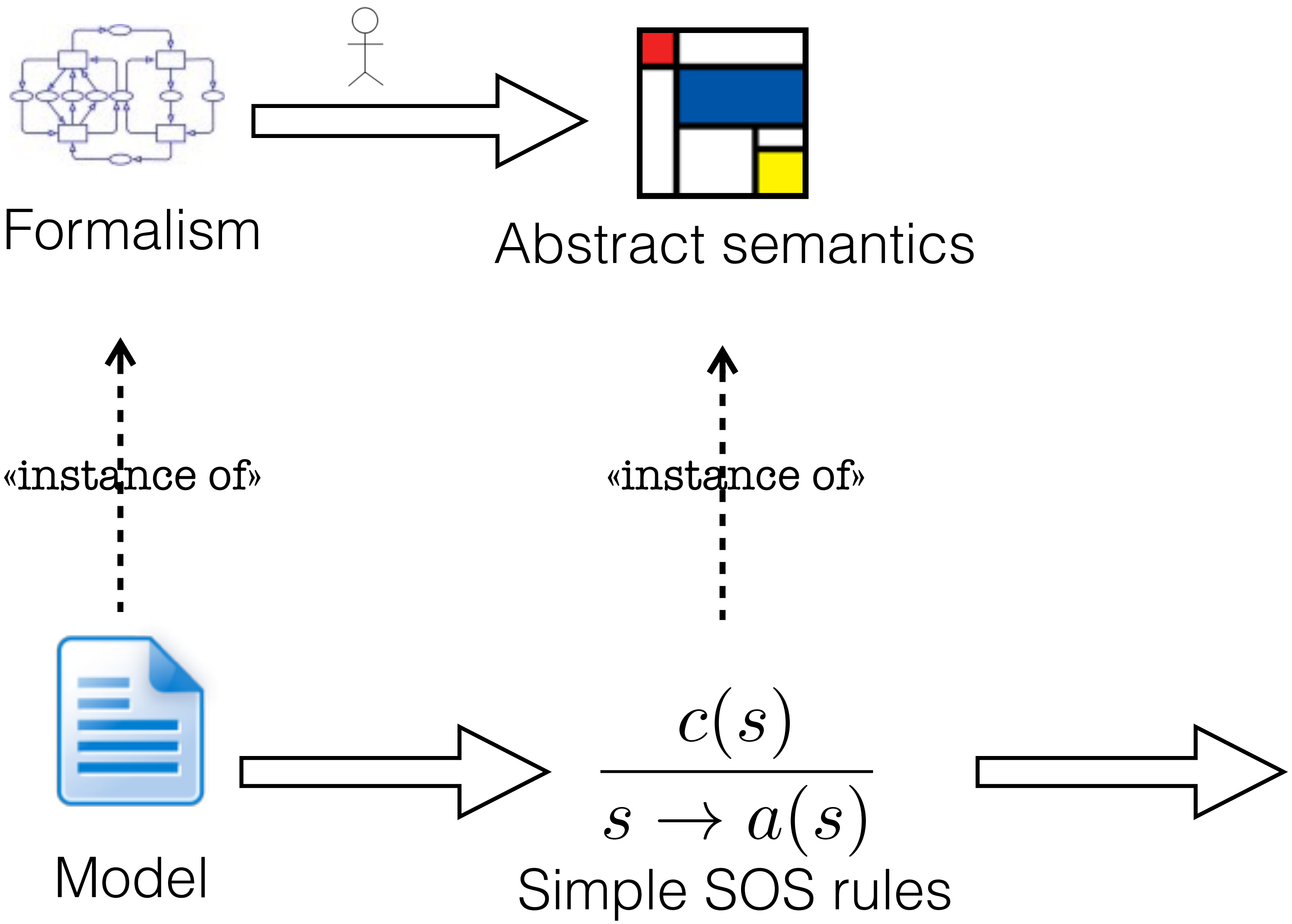
Goal


Formalism
Abstract semantics
«instance of»
«instance of»
Model
c(s)
s → a(s)
Simple SOS rules

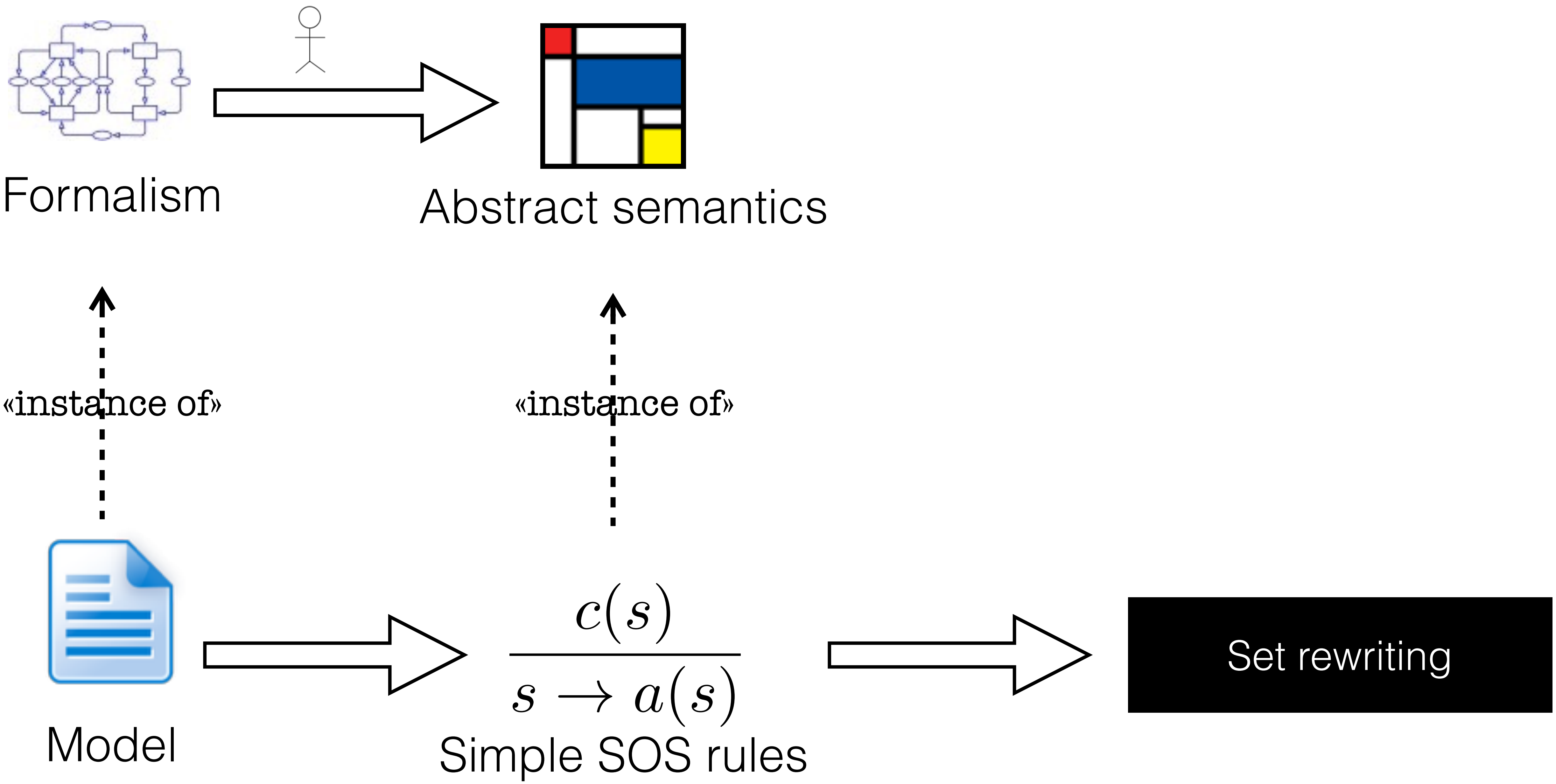
Goal


Formalism
Abstract semantics
«instance of»
«instance of»
Model
$\frac{c(s)}{s \to a(s)}$
Simple SOS rules

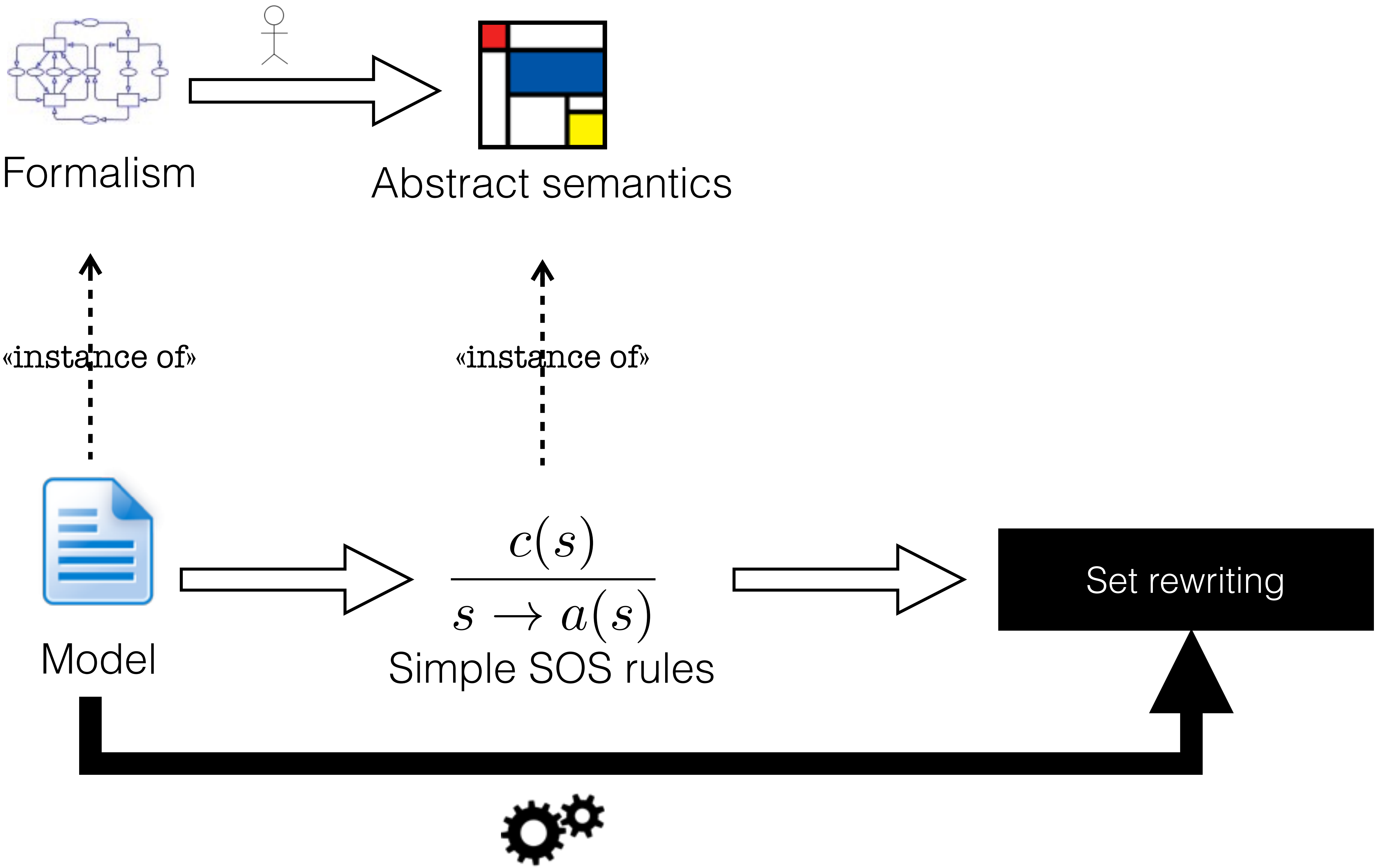
Goal


Formalism
Abstract semantics
«instance of»
«instance of»
Model
$$\frac{c(s)}{s \to a(s)}$$
Simple SOS rules

Goal


Formalism
Abstract semantics
«instance of»
«instance of»
Model
$\frac{c(s)}{s \to a(s)}$
Simple SOS rules
Set rewriting

Goal


Formalism
Abstract semantics
«instance of»
«instance of»
Model
$\frac{c(s)}{s \rightarrow a(s)}$
Simple SOS rules
Set rewriting

The translation

$$\frac{c(s)}{s \to a(s)}$$

⇒

Set rewriting

The translation

$$\frac{c(s)}{s \to a(s)} \quad \Longrightarrow \quad \text{Set rewriting}$$

- stateComp: State $\longrightarrow T_{\Sigma}$

The translation

$$\frac{c(s)}{s \to a(s)} \quad \Longrightarrow \quad \text{Set rewriting}$$

- stateComp: State $\longrightarrow T_{\Sigma}$
- comp: SimpleSOS $\longrightarrow$ Strategies

General principles

$$\frac{c(s)}{s \to a(s)} \quad \Longrightarrow \quad \text{Sequence(comp(c), comp(a))}$$

General principles

$$\frac{c(s)}{s \rightarrow a(s)} \quad \Longrightarrow \quad \text{Sequence(comp(c), comp(a))}$$

- If c is false, comp(c) fails

General principles

$$\frac{c(s)}{s \rightarrow a(s)} \quad \Longrightarrow \quad \text{Sequence(comp(c), comp(a))}$$

- If c is false, comp(c) fails
- comp($c_1 \wedge c_2$) = Sequence(comp(c_1), comp(c_2))

General principles

$$\frac{c(s)}{s \to a(s)} \Longrightarrow \text{Sequence(comp(c), comp(a))}$$

- If c is false, comp(c) fails
- $\text{comp}(c_1 \wedge c_2) = \text{Sequence}(\text{comp}(c_1), \text{comp}(c_2))$
- $\text{comp}(a_1 \circ a_2) = \text{Sequence}(\text{comp}(a_2), \text{comp}(a_1))$

General principles

$$\frac{c(s)}{s \to a(s)} \quad \Longrightarrow \quad \text{Sequence(comp(c), comp(a))}$$

- If c is false, comp(c) fails
- comp($c_1 \wedge c_2$) = Sequence(comp(c_1), comp(c_2))
- comp($a_1 \circ a_2$) = Sequence(comp(a_2), comp(a_1))
- comp(a) = userActComp(a)

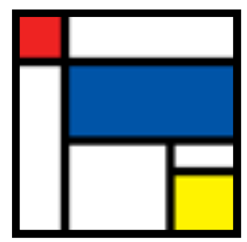
General principles

$$\frac{c(s)}{s \to a(s)} \implies \text{Sequence(comp(c), comp(a))}$$

- If c is false, comp(c) fails
- comp($c_1 \wedge c_2$) = Sequence(comp(c_1), comp(c_2))
- comp($a_1 \circ a_2$) = Sequence(comp(a_2), comp(a_1))
- comp(a) = userActComp(a)
- comp(c) = userPredComp(c)

Case Study: Petri nets

Case Study: Petri nets



Abstract semantics

$$\frac{m \geq in(t)}{m \rightarrow_t m - in(t) + out(t)}$$

Case Study: Petri nets

Abstract semantics

$$\frac{m \geq in(t)}{m \rightarrow_t m - in(t) + out(t)}$$

$$\frac{\bigwedge_{p \in P} test_{p,in(t)(p)}(m) \quad P = \{p_1, \ldots, p_{n'}\}}{m \rightarrow dec_{p_1,in(t)(p_1)} \circ \cdots \circ dec_{p_{n'},in(t)(p_{n'})} \circ inc_{p_1,in(t)(p_1)} \circ \cdots \circ inc_{p_{n'},in(t)(p_{n'})}(m)}$$

Case Study: Petri nets

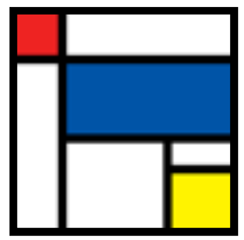
Abstract semantics

$$\frac{m \geq in(t)}{m \to_t m - in(t) + out(t)}$$

$$\frac{\bigwedge_{p \in P} test_{p,in(t)(p)}(m) \quad P = \{p_1, \ldots, p_{n'}\}}{m \to dec_{p_1,in(t)(p_1)} \circ \cdots \circ dec_{p_{n'},in(t)(p_{n'})} \circ inc_{p_1,in(t)(p_1)} \circ \cdots \circ inc_{p_{n'},in(t)(p_{n'})}(m)}$$

- userPredComp($test_{p,in(t)}$) = ???

Case Study: Petri nets



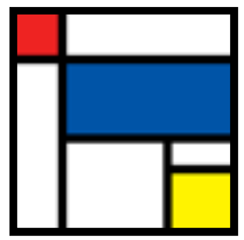
Abstract semantics

$$\frac{m \geq in(t)}{m \rightarrow_t m - in(t) + out(t)}$$

$$\frac{\bigwedge_{p \in P} test_{p,in(t)(p)}(m) \quad P = \{p_1, \ldots, p_{n'}\}}{m \rightarrow dec_{p_1,in(t)(p_1)} \circ \cdots \circ dec_{p_{n'},in(t)(p_{n'})} \circ inc_{p_1,in(t)(p_1)} \circ \cdots \circ inc_{p_{n'},in(t)(p_{n'})}(m)}$$

- userPredComp($test_{p,in(t)}$) = ???
- userActComp($dec_{p,in(t)}$) = ???

Case Study: Petri nets



Abstract semantics

$$\frac{m \geq in(t)}{m \rightarrow_t m - in(t) + out(t)}$$

$$\frac{\bigwedge_{p \in P} test_{p,in(t)(p)}(m) \quad P = \{p_1, \ldots, p_{n'}\}}{m \rightarrow dec_{p_1,in(t)(p_1)} \circ \cdots \circ dec_{p_{n'},in(t)(p_{n'})} \circ inc_{p_1,in(t)(p_1)} \circ \cdots \circ inc_{p_{n'},in(t)(p_{n'})}(m)}$$

- userPredComp($\text{test}_{\text{p,in(t)}}$) = ???
- userActComp($\text{dec}_{\text{p,in(t)}}$) = ???
- userActComp($\text{inc}_{\text{p,in(t)}}$) = ???

State Space

State Space

- Sorts: nat, pname, mapping

State Space

- Sorts: nat, pname, mapping
- zero: nat

State Space

- Sorts: nat, pname, mapping
- zero: nat
- s: nat $\longrightarrow$ nat

State Space

- Sorts: nat, pname, mapping
- zero: nat
- s: nat $\longrightarrow$ nat
- P1, P2, P3: pname

State Space

- Sorts: nat, pname, mapping
- zero: nat
- s: nat $\longrightarrow$ nat
- P1, P2, P3: pname
- empty: mapping

State Space

- Sorts: nat, pname, mapping
- zero: nat
- s: nat ⟶ nat
- P1, P2, P3: pname
- empty: mapping
- map: pname, nat, mapping ⟶ mapping

State Space

- Sorts: nat, pname, mapping
- zero: nat
- s: nat ⟶ nat
- P1, P2, P3: pname
- empty: mapping
- map: pname, nat, mapping ⟶ mapping
- map(P1, 1, map(P2, 0, map(P3, 5, empty)))

$test_{p,in(t)}$

$test_{p,in(t)}$

- $applyTo(C, S) = ITE(C, S, One_3(applyTo(C, S))$

$test_{p,in(t)}$

- $applyTo(C, S) = ITE(C, S, One_3(applyTo(C, S))$
- $checkLoc_p = \{map(p, \$x, \$m) \rightsquigarrow map(p, \$x, \$m)\}$

$\text{test}_{p,in(t)}$

- $\text{applyTo}(C, S) = \text{ITE}(C, S, \text{One}_3(\text{applyTo}(C, S))$
- $\text{checkLoc}_p = \{\text{map}(p, \$x, \$m) \rightsquigarrow \text{map}(p, \$x, \$m)\}$
- $\text{testCond}_n = \{\text{map}(\$p, s(..(s(\$x))), \$m) \rightsquigarrow$
  $\text{map}(\$p, s(..(s(\$x))), \$m)\}$

$test_{p,in(t)}$

- $applyTo(C, S) = ITE(C, S, One_3(applyTo(C, S))$
- $checkLoc_p = \{map(p, \$x, \$m) \rightsquigarrow map(p, \$x, \$m)\}$
- $testCond_n = \{map(\$p, s(..(s(\$x))), \$m) \rightsquigarrow map(\$p, s(..(s(\$x))), \$m)\}$
- $userPredComp(test_{p,n}) = applyTo(checkLoc_p, testCond_n)$

$dec_{p,in(t)}$ & $inc_{p,in(t)}$

$dec_{p,in(t)}$ & $inc_{p,in(t)}$

- $decStrat_n$ = { map($p, s(..(s($x))), $m) ⇝
  map($p, $x, $m) }

$\mathrm{dec}_{p,in(t)}$ & $\mathrm{inc}_{p,in(t)}$

- $\mathrm{decStrat}_n$ = { map($p, s(..(s($x))), $m) ⤳
  map($p, $x, $m) }

- $\mathrm{incStrat}_n$ = { map($p, $x, $m) ⤳
  map($p, s(..(s($x))), $m)}

$dec_{p,in(t)}$ & $inc_{p,in(t)}$

- $decStrat_n$ = { map($p, s(..(s($x))), $m) ⤳
  map($p, $x, $m) }

- $incStrat_n$ = { map($p, $x, $m) ⤳
  map($p, s(..(s($x))), $m)}

- userActComp($dec_{p,n}$) =
  applyTo($checkLoc_p$, $decStrat_n$)

$\mathrm{dec}_{p,in(t)}$ & $\mathrm{inc}_{p,in(t)}$

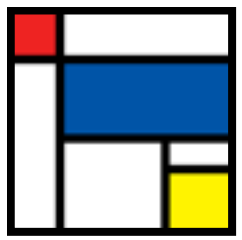
- $\mathrm{decStrat}_n$ = { map($p, s(..(s($x))), $m) ⤳
  map($p, $x, $m) }

- $\mathrm{incStrat}_n$ = { map($p, $x, $m) ⤳
  map($p, s(..(s($x))), $m)}

- userActComp($\mathrm{dec}_{p,n}$) =
  applyTo($\mathrm{checkLoc}_p$, $\mathrm{decStrat}_n$)

- userActComp($\mathrm{inc}_{p,n}$) =
  applyTo($\mathrm{checkLoc}_p$, $\mathrm{incStrat}_n$)

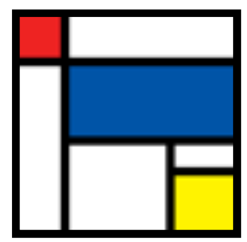
Case Study: Divine



Abstract semantics

$$\frac{c_1 \wedge c_2 \wedge \ldots \wedge c_n}{m \rightarrow a_1 \circ \ldots \circ a_m(m)}$$

Case Study: Divine

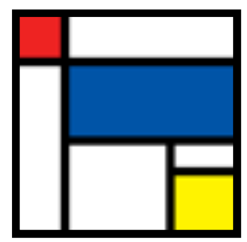


Abstract semantics

$$\frac{c_1 \wedge c_2 \wedge \ldots \wedge c_n}{m \rightarrow a_1 \circ \ldots \circ a_m(m)}$$

- SPIN-like language

Case Study: Divine

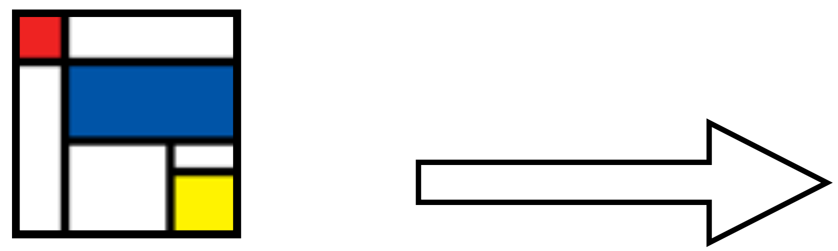


Abstract semantics

$$\frac{c_1 \wedge c_2 \wedge \ldots \wedge c_n}{m \to a_1 \circ \ldots \circ a_m(m)}$$

- SPIN-like language
- Guards, actions and processes

Case Study: Divine

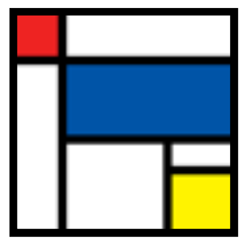


Abstract semantics

$$\frac{c_1 \wedge c_2 \wedge \ldots \wedge c_n}{m \rightarrow a_1 \circ \ldots \circ a_m(m)}$$

- SPIN-like language
- Guards, actions and processes
- c_1, $c_2, \ldots,$ c_n??

Case Study: Divine



Abstract semantics

$$\frac{c_1 \wedge c_2 \wedge \ldots \wedge c_n}{m \rightarrow a_1 \circ \ldots \circ a_m(m)}$$

- SPIN-like language
- Guards, actions and processes
- c_1, c_2,…, c_n??
- a_1, a_2, …, a_m??

State Space

State Space

- M: set of mappings from variables to int

State Space

- M: set of mappings from variables to int
- States = $\{(\text{Stack} \times M) \cup (\text{Guards} \times M)\}$

State Space

- M: set of mappings from variables to int
- States = {(Stack × M) ∪ (Guards × M) }
- test: bool, mapping ⟶ state

State Space

- M: set of mappings from variables to int
- States = {(Stack × M) ∪ (Guards × M) }
- test: bool, mapping ⟶ state
- Stack ops:

State Space

- M: set of mappings from variables to int
- States = {(Stack × M) ∪ (Guards × M) }
- test: bool, mapping ⟶ state
- Stack ops:
  - push(n, s), returns the new stack

State Space

- M: set of mappings from variables to int
- States = {(Stack × M) ∪ (Guards × M) }
- test: bool, mapping ⟶ state
- Stack ops:
    - push(n, s), returns the new stack
    - pop(s) returns the stack without top element

State Space

- M: set of mappings from variables to int
- States = {(Stack × M) ∪ (Guards × M) }
- test: bool, mapping ⟶ state
- Stack ops:
    - push(n, s), returns the new stack
    - pop(s) returns the stack without top element
    - top(s) returns top element of the stack

$$
\begin{aligned}
test &: BoolExpr \times M \to \{true, false\} \\
&: \langle b, \rho \rangle \mapsto \begin{cases} true, \text{ if } b = true \\ false, \text{ otherwise} \end{cases} \\
push_n &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(n, st), \rho \rangle \\
add &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(top(st) + top(pop(st)), pop(pop(st))), \rho \rangle \\
subt &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(top(st) - top(pop(st)), pop(pop(st))), \rho \rangle
\end{aligned}
$$

$$
\begin{aligned}
test &: BoolExpr \times M \rightarrow \{true, false\} \\
&: \langle b, \rho \rangle \mapsto \begin{cases} true, \text{ if } b = true \\ false, \text{ otherwise} \end{cases} \\
push_n &: Stack \times M \rightarrow Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(n, st), \rho \rangle \\
add &: Stack \times M \rightarrow Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(top(st) + top(pop(st)), pop(pop(st))), \rho \rangle \\
subt &: Stack \times M \rightarrow Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(top(st) - top(pop(st)), pop(pop(st))), \rho \rangle \\
read_v &: Stack \times M \rightarrow Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(\rho(v)), \rho \rangle \\
write_v &: Stack \times M \rightarrow Stack \times M
\end{aligned}
$$

$$
\begin{aligned}
test &: BoolExpr \times M \to \{true, false\} \\
&: \langle b, \rho \rangle \mapsto \begin{cases} true, \text{ if } b = true \\ false, \text{ otherwise} \end{cases} \\
push_n &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(n, st), \rho \rangle \\
add &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(top(st) + top(pop(st)), pop(pop(st))), \rho \rangle \\
subt &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(top(st) - top(pop(st)), pop(pop(st))), \rho \rangle \\
read_v &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(\rho(v)), \rho \rangle \\
write_v &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle pop(st), \rho[v = top(st)] \rangle \\
eq &: Stack \times M \to BoolExpr \times M
\end{aligned}
$$

$$
\begin{aligned}
test &: BoolExpr \times M \to \{true, false\} \\
&: \langle b, \rho \rangle \mapsto \begin{cases} true, \text{ if } b = true \\ false, \text{ otherwise} \end{cases} \\
push_n &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(n, st), \rho \rangle \\
add &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(top(st) + top(pop(st)), pop(pop(st))), \rho \rangle \\
subt &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(top(st) - top(pop(st)), pop(pop(st))), \rho \rangle \\
read_v &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(\rho(v)), \rho \rangle \\
write_v &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle pop(st), \rho[v = top(st)] \rangle \\
eq &: Stack \times M \to BoolExpr \times M \\
&: \langle st, \rho \rangle \mapsto \langle top(st) = top(pop(st)), \rho \rangle
\end{aligned}
$$

$$\begin{aligned}
push_n &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(n, st), \rho \rangle \\
add &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(top(st) + top(pop(st)), pop(pop(st))), \rho \rangle \\
subt &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(top(st) - top(pop(st)), pop(pop(st))), \rho \rangle \\
read_v &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(\rho(v)), \rho \rangle \\
write_v &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle pop(st), \rho[v = top(st)] \rangle \\
eq &: Stack \times M \to BoolExpr \times M \\
&: \langle st, \rho \rangle \mapsto \langle top(st) = top(pop(st)), \rho \rangle
\end{aligned}$$

$$
\begin{aligned}
&: \langle st, \rho \rangle \mapsto \langle push(n, st), \rho \rangle \\
add &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(top(st) + top(pop(st)), pop(pop(st))), \rho \rangle \\
subt &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(top(st) - top(pop(st)), pop(pop(st))), \rho \rangle \\
read_v &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(\rho(v)), \rho \rangle \\
write_v &: Stack \times M \to Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle pop(st), \rho[v = top(st)] \rangle \\
eq &: Stack \times M \to BoolExpr \times M \\
&: \langle st, \rho \rangle \mapsto \langle top(st) = top(pop(st)), \rho \rangle
\end{aligned}
$$

$$: \langle st, \rho \rangle \mapsto \langle push(top(st) + top(pop(st)), pop(pop(st))), \rho \rangle$$

$$
\begin{aligned}
subt &: Stack \times M \rightarrow Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(top(st) - top(pop(st)), pop(pop(st))), \rho \rangle \\
read_v &: Stack \times M \rightarrow Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle push(\rho(v)), \rho \rangle \\
write_v &: Stack \times M \rightarrow Stack \times M \\
&: \langle st, \rho \rangle \mapsto \langle pop(st), \rho[v = top(st)] \rangle \\
eq &: Stack \times M \rightarrow BoolExpr \times M \\
&: \langle st, \rho \rangle \mapsto \langle top(st) = top(pop(st)), \rho \rangle
\end{aligned}
$$

Example

- **trans** v’ = 1 + v; v := v + 1

$$\frac{test \circ eq \circ read_{v'} \circ add \circ read_v \circ push_1(\langle st, m \rangle)}{\langle st, m \rangle \rightarrow write_v \circ add \circ read_v \circ push_1(\langle st, m \rangle)}$$

Example

- **trans** v' = 1 + v; v := v + 1

$$\frac{test \circ eq \circ read_{v'} \circ add \circ read_v \circ push_1(\langle st, m \rangle)}{\langle st, m \rangle \rightarrow write_v \circ add \circ read_v \circ push_1(\langle st, m \rangle)}$$

Example

- **trans** v' = 1 + v; v := v + 1

$$\frac{test \circ eq \circ read_{v'} \circ add \circ read_v \circ push_1(\langle st, m \rangle)}{\langle st, m \rangle \to write_v \circ add \circ read_v \circ push_1(\langle st, m \rangle)}$$

Example

- **trans** v' = 1 + v; v := v + 1

$$\frac{test \circ eq \circ read_{v'} \circ add \circ read_v \circ push_1(\langle st, m \rangle)}{\langle st, m \rangle \rightarrow write_v \circ add \circ read_v \circ push_1(\langle st, m \rangle)}$$

Example

- **trans** v' = 1 + v; v := v + 1

$$\frac{test \circ eq \circ read_{v'} \circ add \circ read_v \circ push_1(\langle st, m \rangle)}{\langle st, m \rangle \to write_v \circ add \circ read_v \circ push_1(\langle st, m \rangle)}$$

Translation to Simple SOS Rules

$$
\begin{aligned}
lComp &: DivBasicExpr \rightarrow Cond \cup Act \\
&: c \ \mathbf{and} \ c' \mapsto lComp(c) \wedge lComp(c) \\
&: true \mapsto true \\
&: false \mapsto false \\
&: e = e' \mapsto test \circ eq \circ lComp(e) \circ lComp(e') \\
&: e + e' \mapsto add \circ lComp(e) \circ lComp(e') \\
&: e - e' \mapsto subt \circ lComp(e) \circ lComp(e') \\
&: id := e \mapsto write_v \circ lComp(e) \\
&: a; a \mapsto lComp(a) \circ lComp(a) \\
&: n \mapsto push_n \\
&: v \mapsto read_v
\end{aligned}
$$

$$\frac{test \circ eq \circ read_{v'} \circ add \circ read_v \circ push_1(\langle st, m \rangle)}{\langle st, m \rangle \to write_v \circ add \circ read_v \circ push_1(\langle st, m \rangle)}$$

$$\frac{test \circ eq \circ read_{v'} \circ add \circ read_v \circ push_1(\langle st, m \rangle)}{\langle st, m \rangle \to write_v \circ add \circ read_v \circ push_1(\langle st, m \rangle)}$$

- userPredComp(test) = test(true, \$m) ⇝ \$m

$$\frac{test \circ eq \circ read_{v'} \circ add \circ read_v \circ push_1(\langle st, m\rangle)}{\langle st, m\rangle \rightarrow write_v \circ add \circ read_v \circ push_1(\langle st, m\rangle)}$$

- userPredComp(test) = test(true, $m) ⇝ $m

- equals = { eq(0, 0) ⇝ true,
  eq(s($n1), s($n2)) ⇝ eq ($n1, $n2),
  eq(0, s($n1)) ⇝ false,
  eq(s($n1), 0) ⇝ false }

$$\frac{test \circ eq \circ read_{v'} \circ add \circ read_v \circ push_1(\langle st, m \rangle)}{\langle st, m \rangle \rightarrow write_v \circ add \circ read_v \circ push_1(\langle st, m \rangle)}$$

- userPredComp(test) = test(true, $m) ⇝ $m
- equals = { eq(0, 0) ⇝ true,
  eq(s($n1), s($n2)) ⇝ eq ($n1, $n2),
  eq(0, s($n1)) ⇝ false,
  eq(s($n1), 0) ⇝ false }
- RewriteSet(S) = Choice(Union(S, Not(S)),
  Choice(S, Not(S)))

$$\frac{test \circ eq \circ read_{v'} \circ add \circ read_{v} \circ push_1(\langle st, m \rangle)}{\langle st, m \rangle \to write_{v} \circ add \circ read_{v} \circ push_1(\langle st, m \rangle)}$$

- userPredComp(test) = test(true, $m) ⇝ $m

- equals = { eq(0, 0) ⇝ true,
  eq(s($n1), s($n2)) ⇝ eq ($n1, $n2),
  eq(0, s($n1)) ⇝ false,
  eq(s($n1), 0) ⇝ false }

- RewriteSet(S) = Choice(Union(S, Not(S)),
  Choice(S, Not(S)))

- applyEquals = One_1(Fixpoint(RewriteSet(equals)))

readV

```
checkV = map(j, $n, $s) ⇝ map(j, $n, $s)
findVAndApply(S) = ITE(checkV, S,
                       One3(findAndApply(S)))
upSwap = map($v, $n, map(stackH, $n, $s)) ⇝
          map(stackH, $n, map($v, $n, $s))
upAux = Choice(upSwap,
         Sequence(One3(upAux), upSwap))
endUp = map(stackH, $n, map($v, $n, $s)) ⇝
          map(t, $n, map($v, $n, $s))
up = Choice(endUp, upAux)
copy = map(j, $n, $p) ⇝ map(stackElt,
         $n, map(j, $n, $p))
readV = Sequence(findVAndApply(copy), up)
```

Practical results

Kanban problem

Practical results

Kanban problem

- Small Petri net

Practical results

Kanban problem

- Small Petri net
- 16 places & 16 transitions, marking changes with scale parameter

Practical results

Kanban problem

- Small Petri net
- 16 places & 16 transitions, marking changes with scale parameter
- State space for scale parameter 100

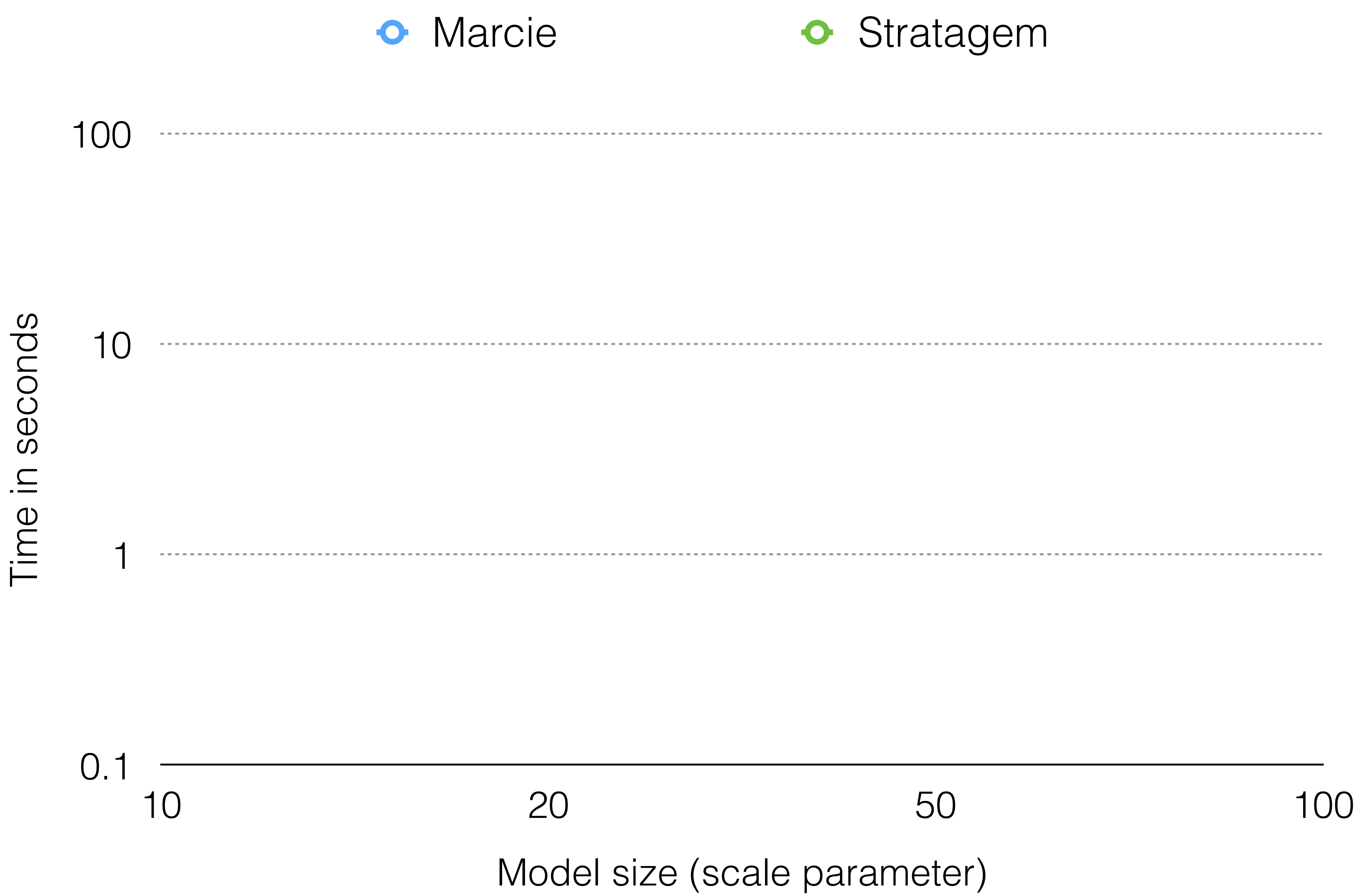
Practical results

Kanban problem

- Small Petri net
- 16 places & 16 transitions, marking changes with scale parameter
- State space for scale parameter 100
  - $1.7263 \cdot 10^{19}$ states

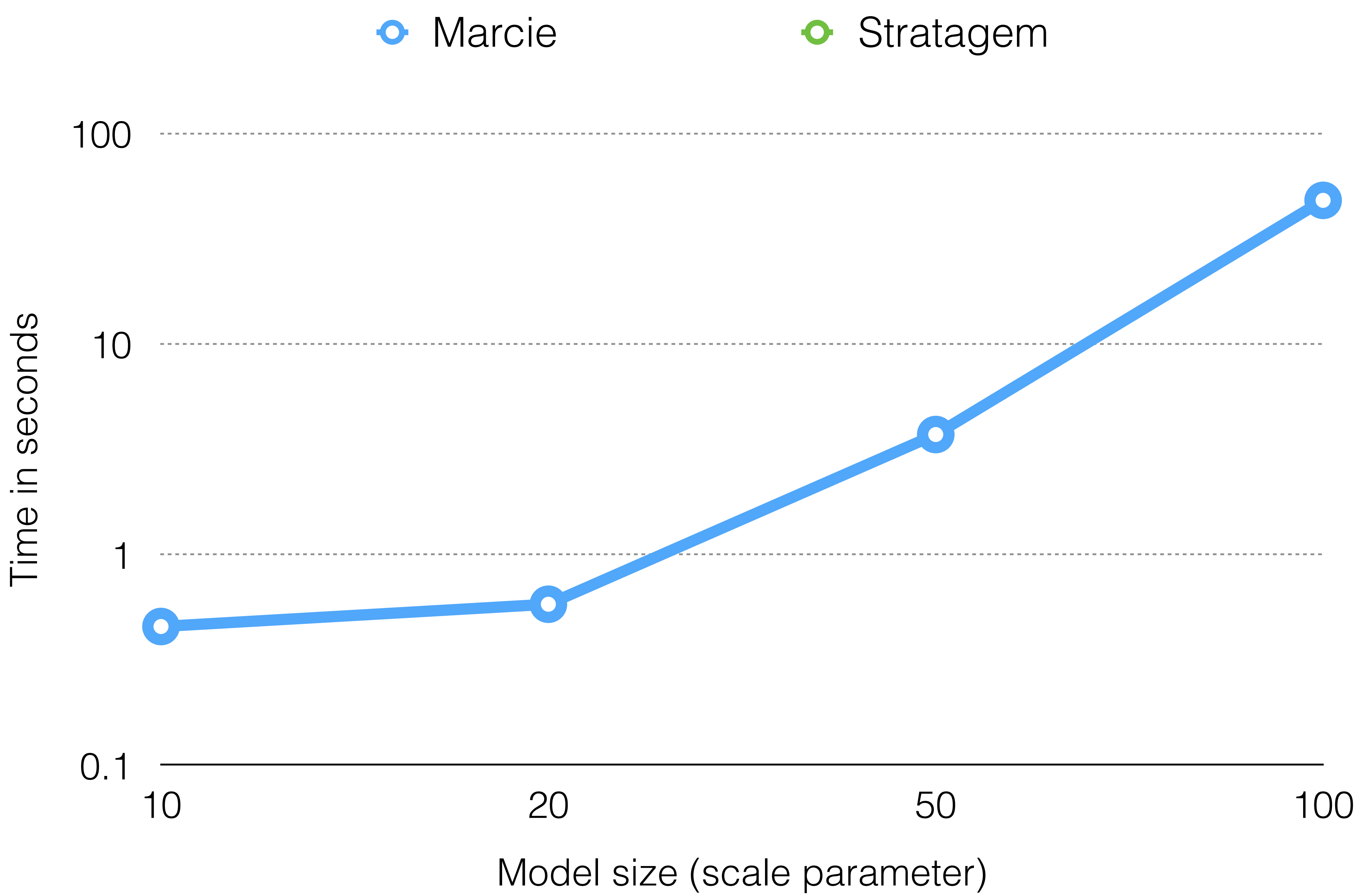
Practical results

Kanban problem

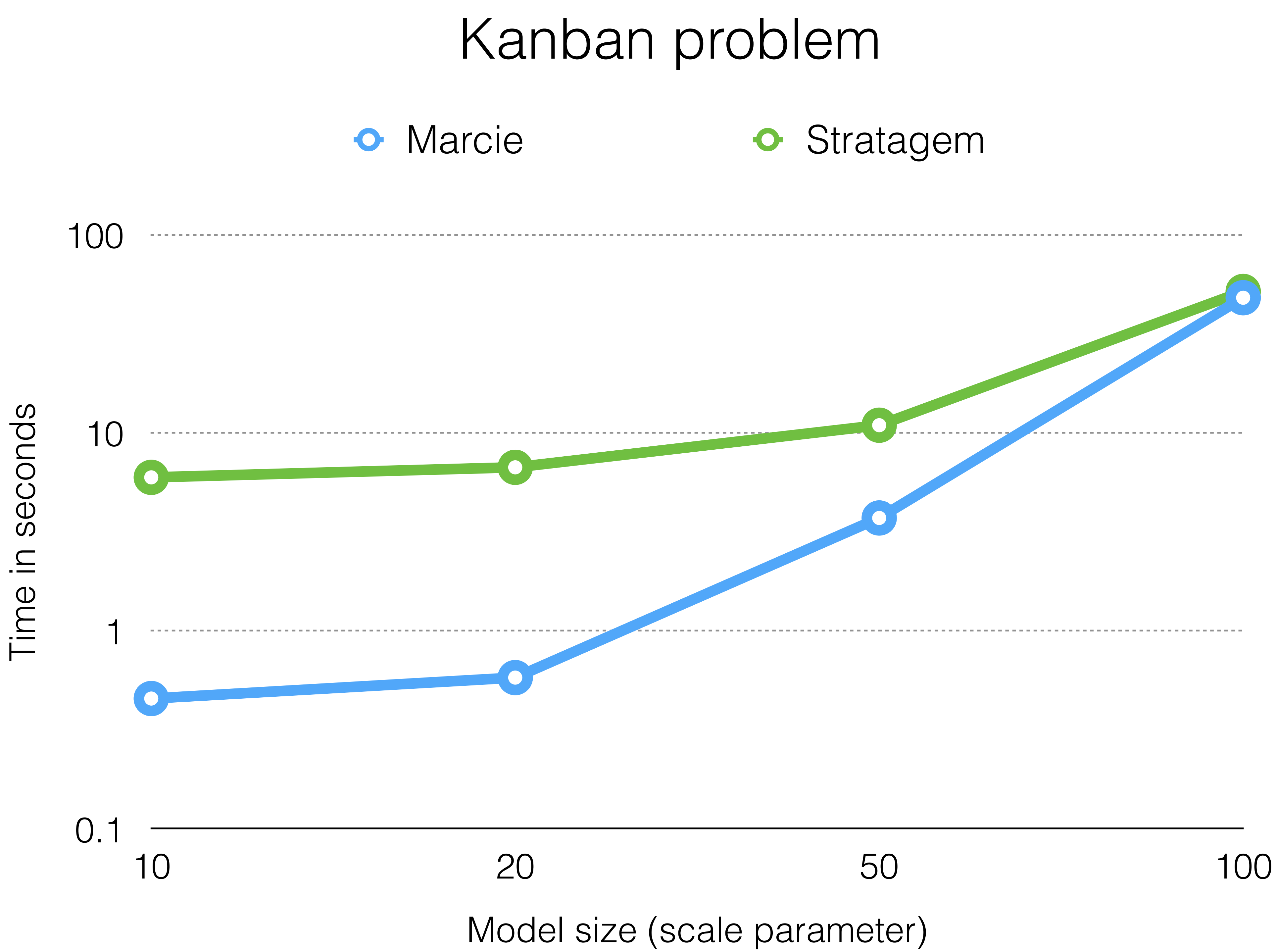


Practical results

Kanban problem



Practical results



Practical results

Sharedmem problem

Practical results

Sharedmem problem

- Petri net’s places and transition increase with scale parameter

Practical results

Sharedmem problem

- Petri net’s places and transition increase with scale parameter
- 2651 places & 5050 transitions for scale parameter 50

Practical results

Sharedmem problem

- Petri net's places and transition increase with scale parameter
- 2651 places & 5050 transitions for scale parameter 50
- State space for scale parameter 50

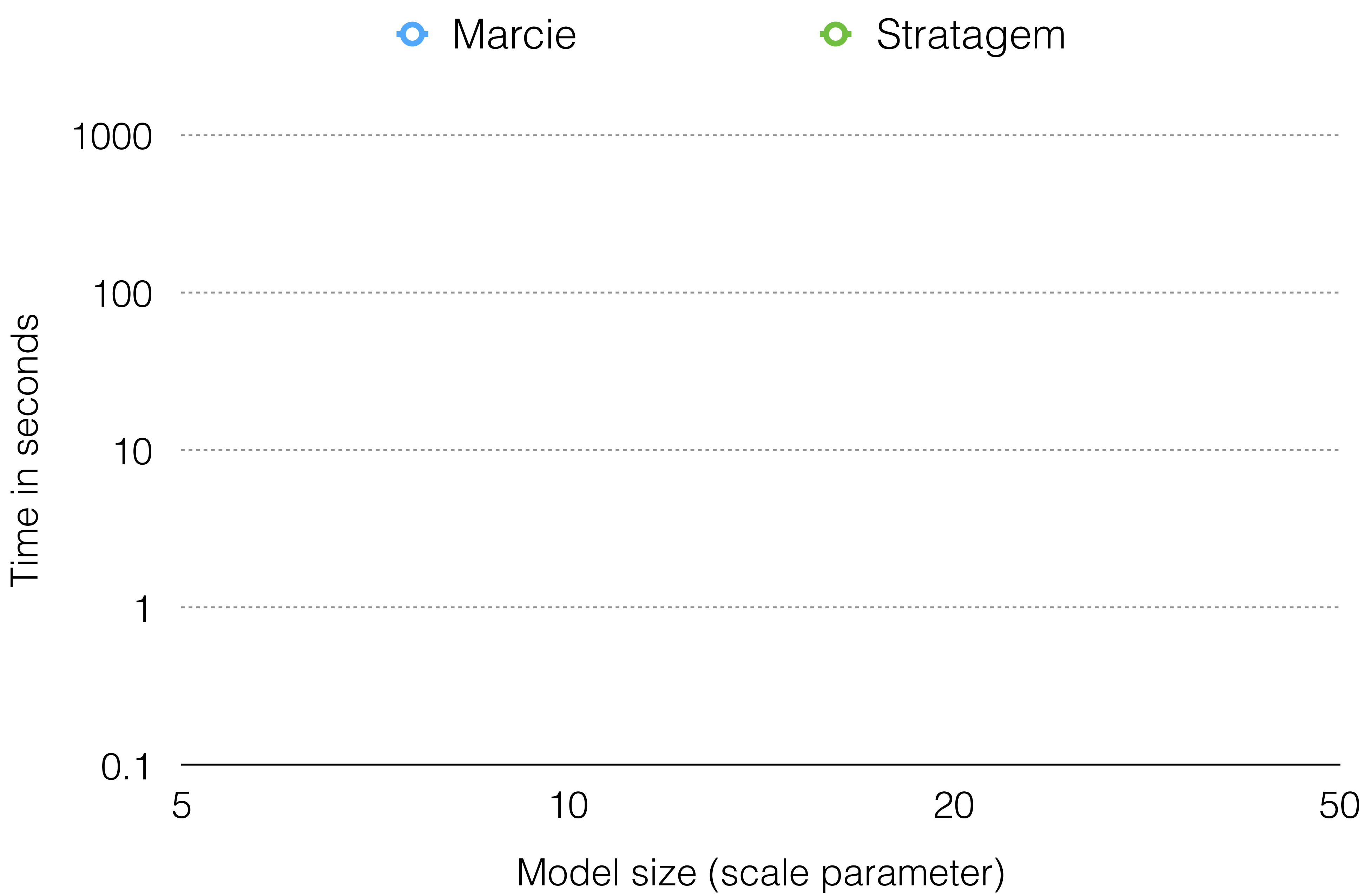
Practical results

Sharedmem problem

- Petri net's places and transition increase with scale parameter
- 2651 places & 5050 transitions for scale parameter 50
- State space for scale parameter 50
    - $5.87 \cdot 10^{26}$ states

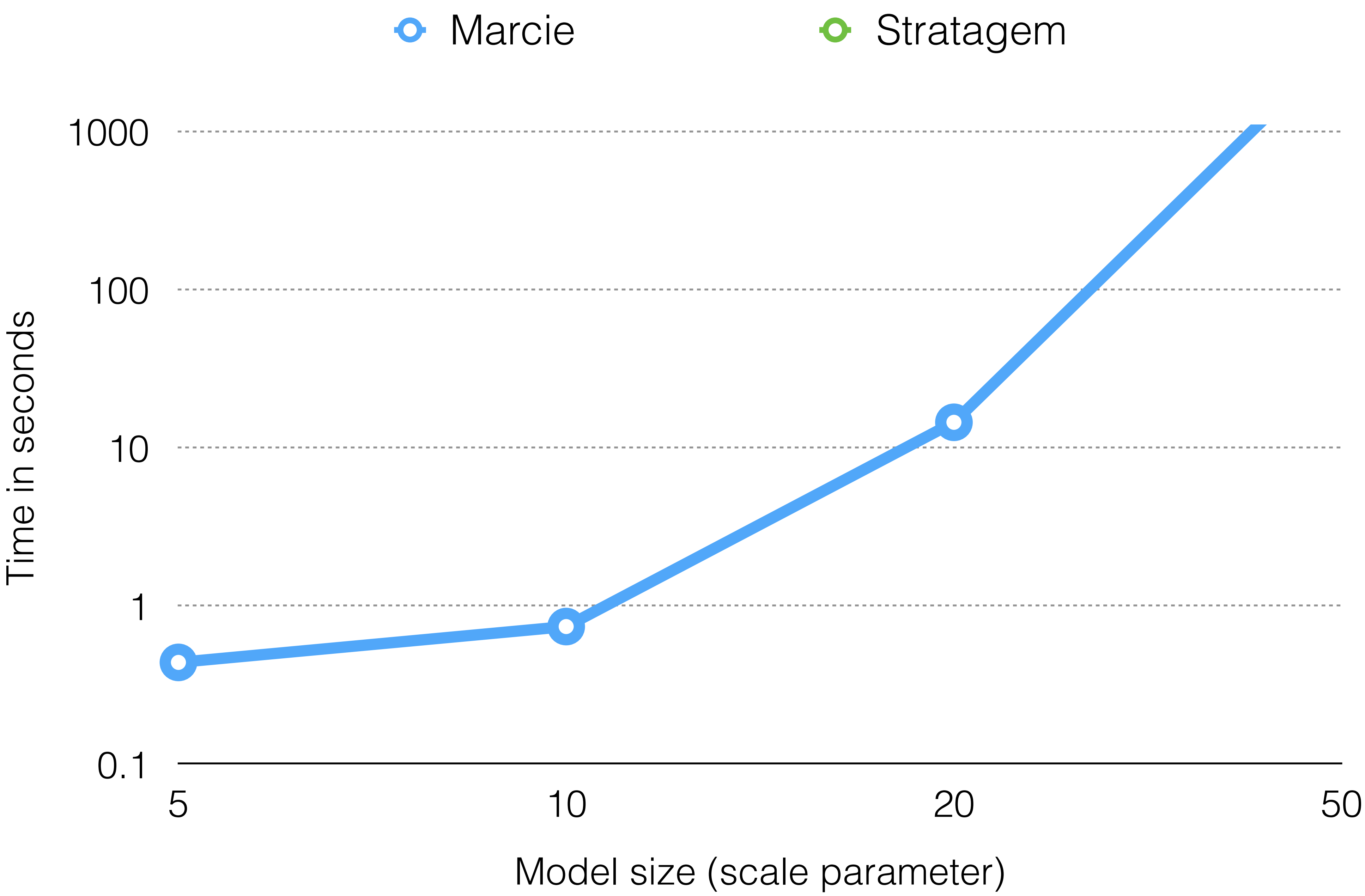
Practical results

SharedMem problem

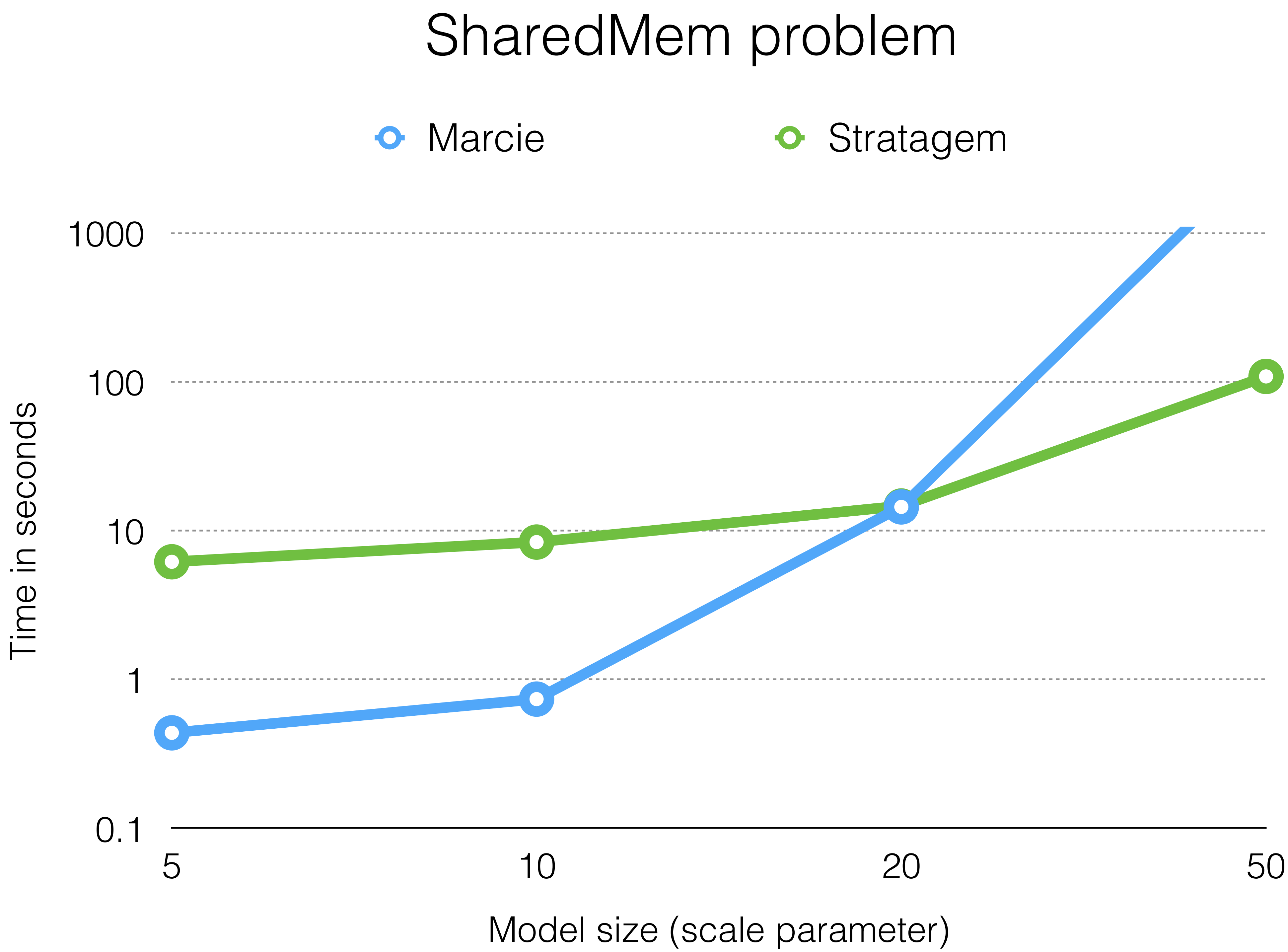


Practical results

SharedMem problem



Practical results



Set rewriting

Saturation: For connaisseurs

Set rewriting

Saturation: For connaisseurs

- Well known DD optimization technique

Set rewriting

Saturation: For connaisseurs

- Well known DD optimization technique
  - Apply local fixpoint in order to reduce peak effect

Set rewriting

Saturation: For connaisseurs

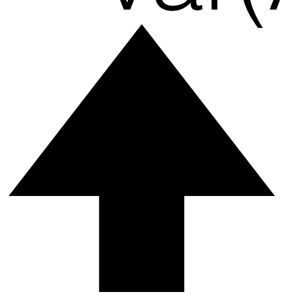
- Well known DD optimization technique
  - Apply local fixpoint in order to reduce peak effect
- $Sat_n(S)$ =
  $Sequence(Choice(One_n(Sat_n(S)),FixPoint(S)), Fixpoint(S))$

Set rewriting

Saturation: For connaisseurs

- Well known DD optimization technique
  - Apply local fixpoint in order to reduce peak effect
- Sat_n(S) =
  Sequence(Choice(One_n(Sat_n(S)),FixPoint(S)), Fixpoint(S))

var(A, 1, var(B, 2, var(C, 0, empty)))

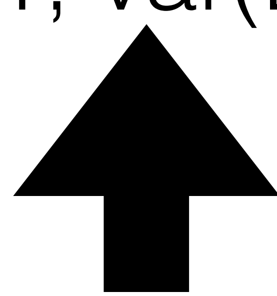


Set rewriting

Saturation: For connaisseurs

- Well known DD optimization technique
  - Apply local fixpoint in order to reduce peak effect
- $Sat_n(S)$ =
  Sequence(Choice($One_n(Sat_n(S))$,FixPoint(S)), Fixpoint(S))

var(A, 1, var(B, 2, var(C, 0, empty)))



Set rewriting

Saturation: For connaisseurs

- Well known DD optimization technique
  - Apply local fixpoint in order to reduce peak effect
- $Sat_n(S)$ =
  Sequence(Choice($One_n(Sat_n(S))$,FixPoint(S)), Fixpoint(S))

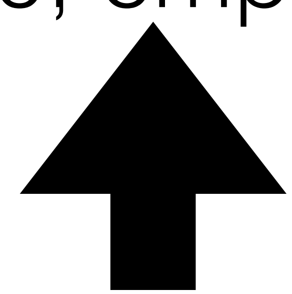
var(A, 1, var(B, 2, var(C, 0, empty)))

Set rewriting

Saturation: For connaisseurs

- Well known DD optimization technique
  - Apply local fixpoint in order to reduce peak effect
- $Sat_n(S)$ =
  Sequence(Choice(One_n(Sat_n(S)),FixPoint(S)), Fixpoint(S))

var(A, 1, var(B, 2, var(C, 0, empty)))



Set rewriting

Saturation: For connaisseurs

- Well known DD optimization technique
  - Apply local fixpoint in order to reduce peak effect
- $Sat_n(S)$ =
  Sequence(Choice(One_n(Sat_n(S)),FixPoint(S)), Fixpoint(S))

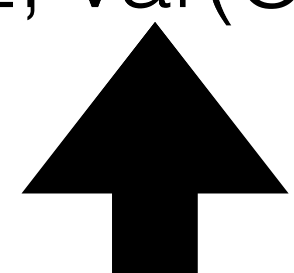
var(A, 1, var(B, 2, var(C, 0, empty)))

Set rewriting

Saturation: For connaisseurs

- Well known DD optimization technique
  - Apply local fixpoint in order to reduce peak effect
- $Sat_n(S)$ =
  Sequence(Choice($One_n(Sat_n(S))$,FixPoint(S)), Fixpoint(S))

var(A, 1, var(B, 2, var(C, 0, empty)))



Set rewriting

Saturation: For connaisseurs

- Well known DD optimization technique
  - Apply local fixpoint in order to reduce peak effect
- $Sat_n(S)$ =
  Sequence(Choice(One_n(Sat_n(S)),FixPoint(S)), Fixpoint(S))

var(A, 1, var(B, 2, var(C, 0, empty)))

Set rewriting

Saturation: For connaisseurs

- Well known DD optimization technique
  - Apply local fixpoint in order to reduce peak effect
- $Sat_n(S)$ =
  Sequence(Choice(One_n(Sat_n(S)),FixPoint(S)), Fixpoint(S))

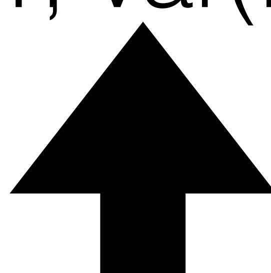
var(A, 1, var(B, 2, var(C, 0, empty)))

Set rewriting

Saturation: For connaisseurs

- Well known DD optimization technique
  - Apply local fixpoint in order to reduce peak effect
- $Sat_n(S)$ =
  Sequence(Choice(One_n(Sat_n(S)),FixPoint(S)), Fixpoint(S))

var(A, 1, var(B, 2, var(C, 0, empty)))

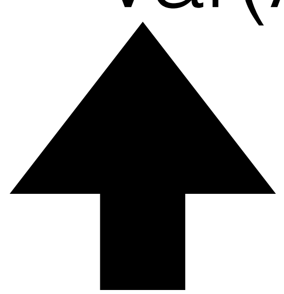


Set rewriting

Saturation: For connaisseurs

- Well known DD optimization technique
    - Apply local fixpoint in order to reduce peak effect
- Sat_n(S) =
  Sequence(Choice(One_n(Sat_n(S)),FixPoint(S)), Fixpoint(S))

var(A, 1, var(B, 2, var(C, 0, empty)))



Set rewriting

Saturation: For connaisseurs

- Well known DD optimization technique
  - Apply local fixpoint in order to reduce peak effect
- Sat_n(S) =
  Sequence(Choice(One_n(Sat_n(S)),FixPoint(S)), Fixpoint(S))

var(A, 1, var(B, 2, var(C, 0, empty)))

Conclusion

- General translation
- Works well with common language constructions
- Efficient implementation