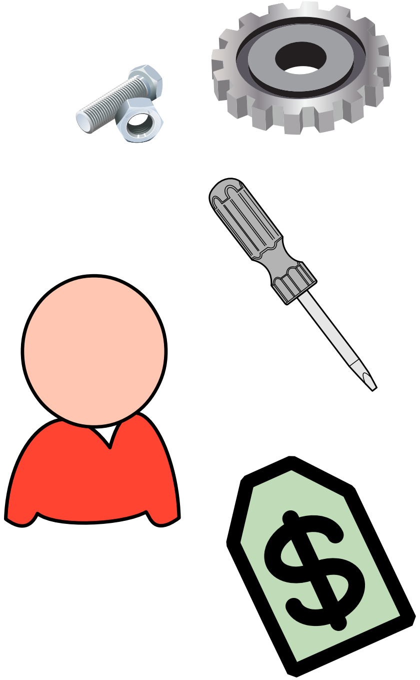
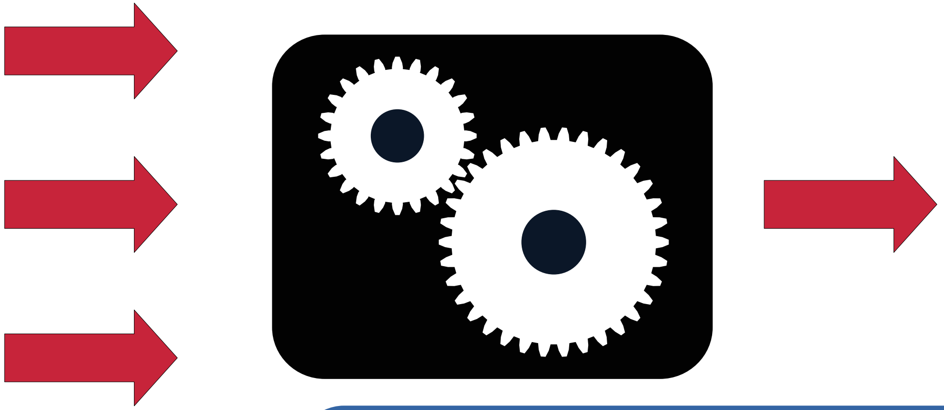
Tool
Accessible Automated Reasoning for Human Robot Collaboration

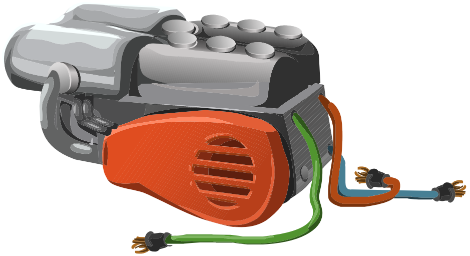
Ortwin Mailahn, Richard Peifer, Rainer Müller ZeMA

Ivan Gavran, *Damien Zufferey* MPI-SWS

Goal







In 10 easy steps...

- User friendly (planning experts)
- Build on top of *off-the-shelf* tools
- Flexible, easy to modify
- Predictable performances

Tool = Tool Ontology and Optimization Language

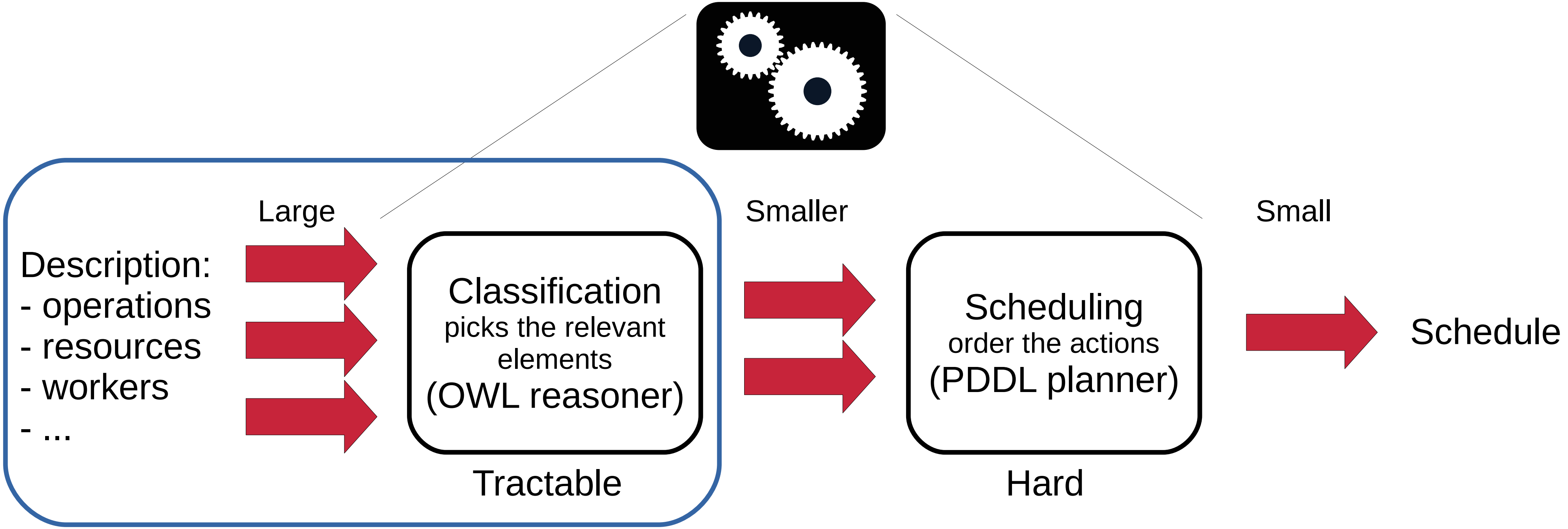
What DSL for planning of assembly systems with human robot cooperation

Why

- Increased flexibility in manufacturing needs more automation in planning
- *Off-the-shelf tools too hard to use for non-expert*

How A DSL that closely match a tractable logic with names matching the target domain to “force” the expert to formulate their problem in way that is nice for constraint solvers.

Under the Hood (Not That Easy)



Tool DSL makes sure this part stays tractable.

Why is a Reasoner needed?

- “Life is complicated”

  There is not a 1-to-1 match between data from multiple sources. For instance, both gluing and welding can attach two objects but they are not always interchangeable.

- Why OWL / Description Logic:

  There is a push in the industry for OWL (Web Ontology Language): Standard Ontologies for Robotics and Automation [IEEE Std 1872-2015]

Expertise for Automated Reasoning

Task	Formulate a small product structure in OWL (No guidance, access to internet resources)
Subject	Employee at ZeMA with CS background (programming but not formal logic)
Result	The product structure expressed in a logic that is 2NEXPTIME… The reasoner was never able to do anything about it (2h timeout).

We tried again with Tool.
The same structure was handled by the reasoner in less than a second.

Tractable Fragments of FOL

First-order logic
(undecidable)

Quantifier-free first order theories:
- Difference logic (cubic)
- Linear programming (P)
- Linear integer programming (NP)

Restriction on quantifiers:
- EPR (NEXPTIME)
- Two-variable logic (2NEXPTIME)

Description logics are roughly fragments of two-variables logic with better complexity.

Description Logic (DL)

FOL	DL
constants	individuals
unary predicate	concept
binary predicate	role

- DL is FOL restricted to [0;2]-ary predicates
- Concept (set) operations: $\sqcap(\cap)$, $\sqsubseteq(\subseteq)$, $\equiv(=)$, $\bot(\emptyset)$
- Quantification: existential/universal restrictions
  - $\exists R.C$ is $\{ a \mid \exists b.\ (a,b) \in R \land C(b) \}$
  - $\forall R.C$ is $\{ a \mid \forall b.\ (a,b) \in R \Rightarrow C(b) \}$
- …

DL used in Tool

SROEL(D)

Data (types and values)

ALC

Attributive language
- Atomic negation
- Concept intersection
- Universal restrictions

Concept negation

role inclusion axioms, reflexivity and irreflexivity, and role disjointness

Nominals (enumerations)

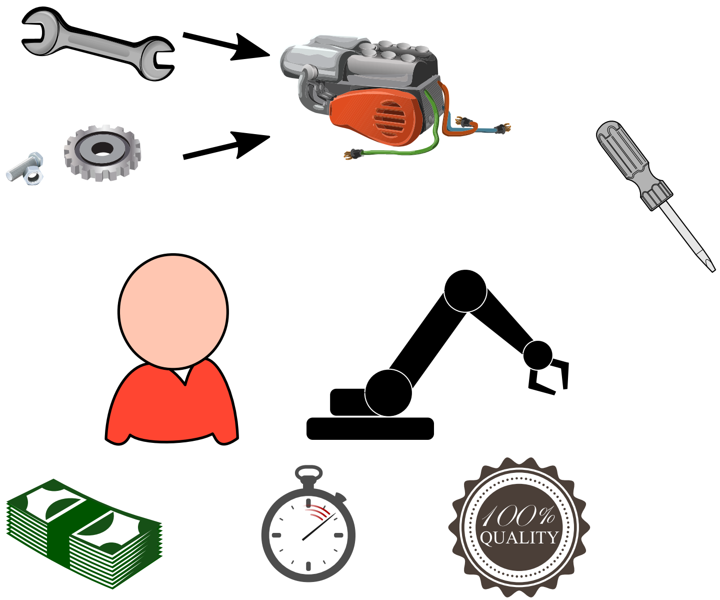
Existential language:
- Concept intersection
- Existential restrictions

Tool Language

name ::=	a string identifier
unit ::=	unit of measurement, e.g., meter
interval ::=	closed, half-open, or open interval of $\mathbb{Q}$
function ::=	PlanningFunction(*name*, *unit*, domain = *name*, range = *interval*)
constraint ::=	(containedIn | disjointFrom) *skill*
skill ::=	Skill(*name*, *constraint**)
|	*function interval constraint**
position ::=	AssemblyPosition(*name*, *skill*, pre = *skill*?, post = *skill*?)
assemblyItem ::=	Component(*name*, weight $\in$ *interval*)
|	AssemblyGroup(*name*, (*position assemblyItem*)*)
actionMatching ::=	MultipleProcess(*assemblyItem*+, *action*+)
action ::=	Operation(*name*, *skill**, *assemblyItem**)
|	Process(*name*, *action**, *assemblyItem**)
|	Process(*name*, *actionMatching*, (*position action*)*)
tool ::=	Tool(*name*, *skill**)
worker ::=	Human(*name*, *skill**, *tool**)
|	Robot(*name*, *skill**, *tool**)
skillMatching ::=	Classification(*action*+, *worker*+, ImportedOntology(*name*)*)
actionCondition ::=	DependsOn(*action*, *action*)
objective ::=	cost $\in \mathbb{Q}$, duration $\in \mathbb{Q}$, capability $\in \{0, \dots, 10\}$, quality $\in \{0, \dots, 10\}$
assessment ::=	Assessment(*objective*, *action*, *worker*?, *tool*?)
|	ActionCostPerHour((*worker* | *tool*) value $\in \mathbb{Q}$)
schedule ::=	Plan(*skillMatching*, *actionCondition**, *assessment**, *objective*)

Skills and Constraints:

```
gluing ⊑ bonding
gluing ⊓ shredding ≡ ⊥
```



Tool Language

name	::=	a string identifier
unit	::=	unit of measurement, e.g., meter
interval	::=	closed, half-open, or open interval of $\mathbb{Q}$
function	::=	PlanningFunction(*name*, *unit*, domain = *name*, range = *interval*)
constraint	::=	(containedIn | disjointFrom) *skill*
skill	::=	Skill(*name*, *constraint**)
	|	*function interval constraint**
position	::=	AssemblyPosition(*name*, *skill*, pre = *skill*?, post = *skill*?)
assemblyItem	::=	Component(*name*, weight $\in$ *interval*)
	|	AssemblyGroup(*name*, (*position assemblyItem*)*)
actionMatching	::=	MultipleProcess(*assemblyItem*+, *action*+)
action	::=	Operation(*name*, *skill**, *assemblyItem**)
	|	Process(*name*, *action**, *assemblyItem**)
	|	Process(*name*, *actionMatching*, (*position action*)*)
tool	::=	Tool(*name*, *skill**)
worker	::=	Human(*name*, *skill**, *tool**)
	|	Robot(*name*, *skill**, *tool**)
skillMatching	::=	Classification(*action*+, *worker*+, ImportedOntology(*name*)*)
actionCondition	::=	DependsOn(*action*, *action*)
objective	::=	cost $\in \mathbb{Q}$, duration $\in \mathbb{Q}$, capability $\in \{0, \dots, 10\}$, quality $\in \{0, \dots, 10\}$
assessment	::=	Assessment(*objective*, *action*, *worker*?, *tool*?)
	|	ActionCostPerHour((*worker* | *tool*) value $\in \mathbb{Q}$)
schedule	::=	Plan(*skillMatching*, *actionCondition**, *assessment**, *objective*)

Skills are capabilities.

Workers and tools provide skills.

Operations require skills.

Constraints:

- between skills

`gluing ⊑ bonding`

`gluing ⊓ shredding ≡ ⊥`

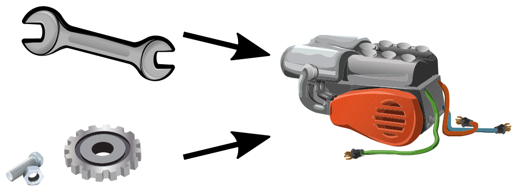
- on skills

lift object up to a certain height

Tool Language: Actions

$$
\begin{array}{rcl}
\mathit{name} & ::= & \text{a string identifier} \\
\mathit{unit} & ::= & \text{unit of measurement, e.g., meter} \\
\mathit{interval} & ::= & \text{closed, half-open, or open interval of } \mathbb{Q} \\
\mathit{function} & ::= & \mathrm{PlanningFunction}(\mathit{name},\ \mathit{unit},\ \mathrm{domain} = \mathit{name},\ \mathrm{range} = \mathit{interval}) \\
\mathit{constraint} & ::= & (\mathrm{containedIn} \mid \mathrm{disjointFrom})\ \mathit{skill} \\
\mathit{skill} & ::= & \mathrm{Skill}(\mathit{name},\ \mathit{constraint}^*) \\
 & \mid & \mathit{function}\ \mathit{interval}\ \mathit{constraint}^* \\
\mathit{position} & ::= & \mathrm{AssemblyPosition}(\mathit{name},\ \mathit{skill},\ \mathrm{pre} = \mathit{skill}?,\ \mathrm{post} = \mathit{skill}?) \\
\mathit{assemblyItem} & ::= & \mathrm{Component}(\mathit{name},\ \mathrm{weight} \in \mathit{interval}) \\
 & \mid & \mathrm{AssemblyGroup}(\mathit{name},\ (\mathit{position}\ \mathit{assemblyItem})^*) \\
\mathit{actionMatching} & ::= & \mathrm{MultipleProcess}(\mathit{assemblyItem}+,\ \mathit{action}+) \\
\mathit{action} & ::= & \mathrm{Operation}(\mathit{name},\ \mathit{skill}^*,\ \mathit{assemblyItem}^*) \\
 & \mid & \mathrm{Process}(\mathit{name},\ \mathit{action}^*,\ \mathit{assemblyItem}^*) \\
 & \mid & \mathrm{Process}(\mathit{name},\ \mathit{actionMatching},\ (\mathit{position}\ \mathit{action})^*) \\
\mathit{tool} & ::= & \mathrm{Tool}(\mathit{name},\ \mathit{skill}^*) \\
\mathit{worker} & ::= & \mathrm{Human}(\mathit{name},\ \mathit{skill}^*,\ \mathit{tool}^*) \\
 & \mid & \mathrm{Robot}(\mathit{name},\ \mathit{skill}^*,\ \mathit{tool}^*) \\
\mathit{skillMatching} & ::= & \mathrm{Classification}(\mathit{action}+,\ \mathit{worker}+,\ \mathrm{ImportedOntology}(\mathit{name})^*) \\
\mathit{actionCondition} & ::= & \mathrm{DependsOn}(\mathit{action},\ \mathit{action}) \\
\mathit{objective} & ::= & \mathrm{cost} \in \mathbb{Q},\ \mathrm{duration} \in \mathbb{Q},\ \mathrm{capability} \in \{0, \ldots, 10\},\ \mathrm{quality} \in \{0, \ldots, 10\} \\
\mathit{assessment} & ::= & \mathrm{Assessment}(\mathit{objective},\ \mathit{action},\ \mathit{worker}?,\ \mathit{tool}?) \\
 & \mid & \mathrm{ActionCostPerHour}((\mathit{worker} \mid \mathit{tool})\ \mathrm{value} \in \mathbb{Q}) \\
\mathit{schedule} & ::= & \mathrm{Plan}(\mathit{skillMatching},\ \mathit{actionCondition}^*,\ \mathit{assessment}^*,\ \mathit{objective})
\end{array}
$$

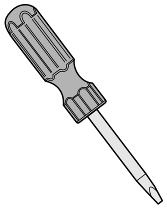
Components at specific positions makes assemblies.



Operations list the skills realize assemblies.

Tool Language: Tools

name	::=	a string identifier
unit	::=	unit of measurement, e.g., meter
interval	::=	closed, half-open, or open interval of $\mathbb{Q}$
function	::=	PlanningFunction(*name*, *unit*, domain = *name*, range = *interval*)
constraint	::=	(containedIn | disjointFrom) *skill*
skill	::=	Skill(*name*, *constraint**)
	|	*function interval constraint**
position	::=	AssemblyPosition(*name*, *skill*, pre = *skill*?, post = *skill*?)
assemblyItem	::=	Component(*name*, weight $\in$ *interval*)
	|	AssemblyGroup(*name*, (*position assemblyItem*)*)
actionMatching	::=	MultipleProcess(*assemblyItem*+, *action*+)
action	::=	Operation(*name*, *skill**, *assemblyItem**)
	|	Process(*name*, *action**, *assemblyItem**)
	|	Process(*name*, *actionMatching*, (*position action*)*)
tool	::=	Tool(*name*, *skill**)
worker	::=	Human(*name*, *skill**, *tool**)
	|	Robot(*name*, *skill**, *tool**)
skillMatching	::=	Classification(*action*+, *worker*+, ImportedOntology(*name*)*)
actionCondition	::=	DependsOn(*action*, *action*)
objective	::=	cost $\in \mathbb{Q}$, duration $\in \mathbb{Q}$, capability $\in \{0, \ldots, 10\}$, quality $\in \{0, \ldots, 10\}$
assessment	::=	Assessment(*objective*, *action*, *worker*?, *tool*?)
	|	ActionCostPerHour((*worker* | *tool*) value $\in \mathbb{Q}$)
schedule	::=	Plan(*skillMatching*, *actionCondition**, *assessment**, *objective*)

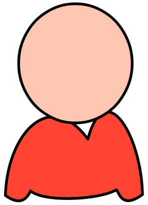



Tools provide skills.

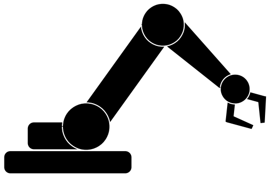
Tool Language: Workers

$$
\begin{array}{rcl}
\mathit{name} & ::= & \text{a string identifier} \\
\mathit{unit} & ::= & \text{unit of measurement, e.g., meter} \\
\mathit{interval} & ::= & \text{closed, half-open, or open interval of } \mathbb{Q} \\
\mathit{function} & ::= & \text{PlanningFunction}(\mathit{name},\ \mathit{unit},\ \text{domain} = \mathit{name},\ \text{range} = \mathit{interval}) \\
\mathit{constraint} & ::= & (\text{containedIn} \mid \text{disjointFrom})\ \mathit{skill} \\
\mathit{skill} & ::= & \text{Skill}(\mathit{name},\ \mathit{constraint}^*) \\
 & \mid & \mathit{function}\ \mathit{interval}\ \mathit{constraint}^* \\
\mathit{position} & ::= & \text{AssemblyPosition}(\mathit{name},\ \mathit{skill},\ \text{pre} = \mathit{skill}?,\ \text{post} = \mathit{skill}?) \\
\mathit{assemblyItem} & ::= & \text{Component}(\mathit{name},\ \text{weight} \in \mathit{interval}) \\
 & \mid & \text{AssemblyGroup}(\mathit{name},\ (\mathit{position}\ \mathit{assemblyItem})^*) \\
\mathit{actionMatching} & ::= & \text{MultipleProcess}(\mathit{assemblyItem}+,\ \mathit{action}+) \\
\mathit{action} & ::= & \text{Operation}(\mathit{name},\ \mathit{skill}^*,\ \mathit{assemblyItem}^*) \\
 & \mid & \text{Process}(\mathit{name},\ \mathit{action}^*,\ \mathit{assemblyItem}^*) \\
 & \mid & \text{Process}(\mathit{name},\ \mathit{actionMatching},\ (\mathit{position}\ \mathit{action})^*) \\
\mathit{tool} & ::= & \text{Tool}(\mathit{name},\ \mathit{skill}^*) \\
\mathit{worker} & ::= & \text{Human}(\mathit{name},\ \mathit{skill}^*,\ \mathit{tool}^*) \\
 & \mid & \text{Robot}(\mathit{name},\ \mathit{skill}^*,\ \mathit{tool}^*) \\
\mathit{skillMatching} & ::= & \text{Classification}(\mathit{action}+,\ \mathit{worker}+,\ \text{ImportedOntology}(\mathit{name})^*) \\
\mathit{actionCondition} & ::= & \text{DependsOn}(\mathit{action},\ \mathit{action}) \\
\mathit{objective} & ::= & \text{cost} \in \mathbb{Q},\ \text{duration} \in \mathbb{Q},\ \text{capability} \in \{0, \ldots, 10\},\ \text{quality} \in \{0, \ldots, 10\} \\
\mathit{assessment} & ::= & \text{Assessment}(\mathit{objective},\ \mathit{action},\ \mathit{worker}?,\ \mathit{tool}?) \\
 & \mid & \text{ActionCostPerHour}((\mathit{worker} \mid \mathit{tool})\ \text{value} \in \mathbb{Q}) \\
\mathit{schedule} & ::= & \text{Plan}(\mathit{skillMatching},\ \mathit{actionCondition}^*,\ \mathit{assessment}^*,\ \mathit{objective})
\end{array}
$$

Human



Robot



Workers have their own skills and acquires the skill provided by their tools.

Tool Language: Cost Functions

name	::=	a string identifier
unit	::=	unit of measurement, e.g., meter
interval	::=	closed, half-open, or open interval of $\mathbb{Q}$
function	::=	PlanningFunction(*name*, *unit*, domain = *name*, range = *interval*)
constraint	::=	(containedIn | disjointFrom) *skill*
skill	::=	Skill(*name*, *constraint**)
	|	*function interval constraint**
position	::=	AssemblyPosition(*name*, *skill*, pre = *skill*?, post = *skill*?)
assemblyItem	::=	Component(*name*, weight $\in$ *interval*)
	|	AssemblyGroup(*name*, (*position assemblyItem*)*)
actionMatching	::=	MultipleProcess(*assemblyItem*+, *action*+)
action	::=	Operation(*name*, *skill**, *assemblyItem**)
	|	Process(*name*, *action**, *assemblyItem**)
	|	Process(*name*, *actionMatching*, (*position action*)*)
tool	::=	Tool(*name*, *skill**)
worker	::=	Human(*name*, *skill**, *tool**)
	|	Robot(*name*, *skill**, *tool**)
skillMatching	::=	Classification(*action*+, *worker*+, ImportedOntology(*name*)*)
actionCondition	::=	DependsOn(*action*, *action*)
objective	::=	cost $\in \mathbb{Q}$, duration $\in \mathbb{Q}$, capability $\in \{0, \ldots, 10\}$, quality $\in \{0, \ldots, 10\}$
assessment	::=	Assessment(*objective*, *action*, *worker*?, *tool*?)
	|	ActionCostPerHour((*worker* | *tool*) value $\in \mathbb{Q}$)
schedule	::=	Plan(*skillMatching*, *actionCondition**, *assessment**, *objective*)

Tool Implementation

```
val factory = PlanningFactory("AircraftFuselage")
val bonding = factory.skill("bonding")
val gluing = factory.skill(name = "gluing",
        containedInAny = setOf(bonding))
val clip = factory.component("clip")
val gluing_clip = factory.operation("gluing a clip",
        treats = setOf(clip),
        requiresSkills = listOf(gluing))
 val assembleClips = factory.process(
    name = "assemble clips",
    subActions = setOf(gluing_clip totalCopies 16))
 val robot1 = factory.robot(name = "robot1",
        skills = listOf(gluing))
```

- As a shallow DSL embedded in Kotlin
- HermiT as OWL reasoner
- LPG-TD as PDDL planner

DL Encoding

Mostly straightforward:

```
val factory = PlanningFactory("AircraftFuselage")
val bonding = factory.skill("bonding")
val gluing = factory.skill(name = "gluing",
        containedInAny = setOf(bonding))
val clip = factory.component("clip")
val gluing_clip = factory.operation("gluing a clip",
        treats = setOf(clip),
        requiresSkills = listOf(gluing))
val assembleClips = factory.process(
    name = "assemble clips",
    subActions = setOf(gluing_clip totalCopies 16))
val robot1 = factory.robot(name = "robot1",
        skills = listOf(gluing))
```

Introduce concept

Introduce concepts and constraint:

$\texttt{gluing} \sqsubseteq \texttt{bonding}$

Introduce concepts and constraint:

$\texttt{skillOf\_robot1} \times \texttt{robot1} \sqsubseteq \texttt{canBeHandledByWorker}$

$\texttt{gluing} \sqsubseteq \texttt{skillOf\_robot1}$

`CanBeHandledByWorker` is a role between skills and workers.

Cardinality Constraints

```
val assembleClips = factory.process(
    name = "assemble clips",
    subActions = setOf(gluing_clip totalCopies 16))
```

- Cardinality constraints combined with combined with other axioms is untractable.
- Sometime is cardinality constraints can be avoided altogether:
  - Instead we introduce 16 constants / individuals.
  - This effectively gives a lower bound of 16 (but no upper bound).

Data Constraints

- Skills can range over values

  lift an object up to 1m.

- Rational function restrictions $\exists F.J$ is $\{ a \mid F(a) \in J \}$
  - F is a data property (function), e.g., height
  - J is an interval in $\mathbb{Q}$, e.g., [0,1]
- $\exists F.J$ is more powerful than $\exists R.C$ because F is a *function*.
- Easily untractable:

  $C \equiv A \sqcup B \quad \Leftrightarrow \quad C \equiv \exists F.[x,z] \;\wedge\; A \equiv \exists F.[x, y] \;\wedge\; B \equiv \exists F.(y, z]$

  Adding concept unions ($\sqcup$) push the DL above NP (ExpTime)

Convex Under-approximation

- Problem: not *convex* anymore

$\exists F.(-\infty,1] \sqsubseteq \exists F.(-\infty, 0] \sqcup \exists F.(0, \infty)$ holds

$\exists F.(-\infty,1] \sqsubseteq \exists F.(-\infty, 0]$ does not hold

$\exists F.(-\infty,1] \sqsubseteq \exists F.(0, \infty)$ does not hold

- Solution: only allows convex deductions
  - Prevent the solver from deducing $\exists F.(-\infty,1] \sqsubseteq \exists F.(-\infty, 0] \sqcup \exists F.(0, \infty)$
  - *Incomplete reasoning*, we can lose solutions.

Planning

- Typical planning problem:
  - Choose which worker/tool will perform an action
  - Respect operation’s dependencies
  - Optimize cost/time/quality
- Tool specific:

  On top of the problem’s element, generate actions with their own cost for tool change, etc.

Experiment Setting

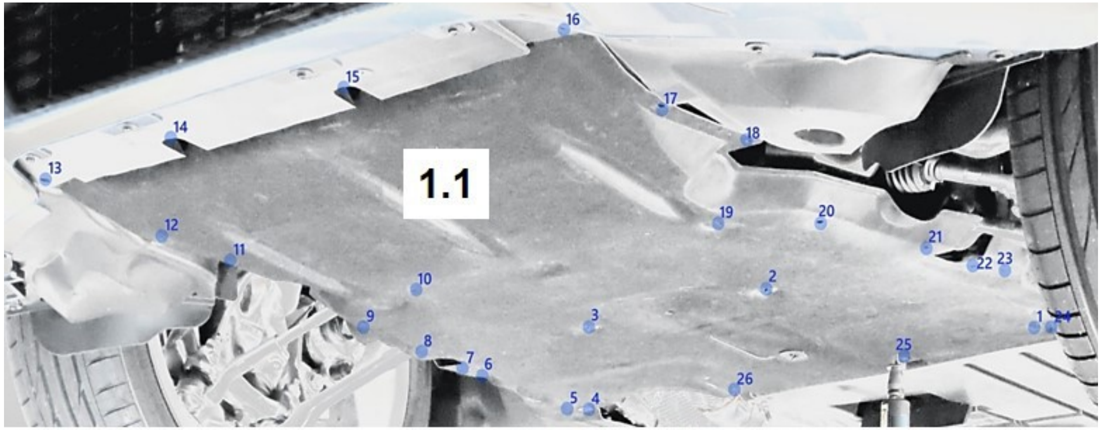
Subject

- Planning expert (Ortwin), but not a computer scientist
- No prior experience in using automated reasoner
- Did learn the basic of Kotlin programming

Effort

- Prior analysis of the planning problem not counted (days-weeks)
- 2/3 of the time for the initial Tool formalization
- 1/3 of the time iterating to get the desired result

Case Study: Car Underbody Panel



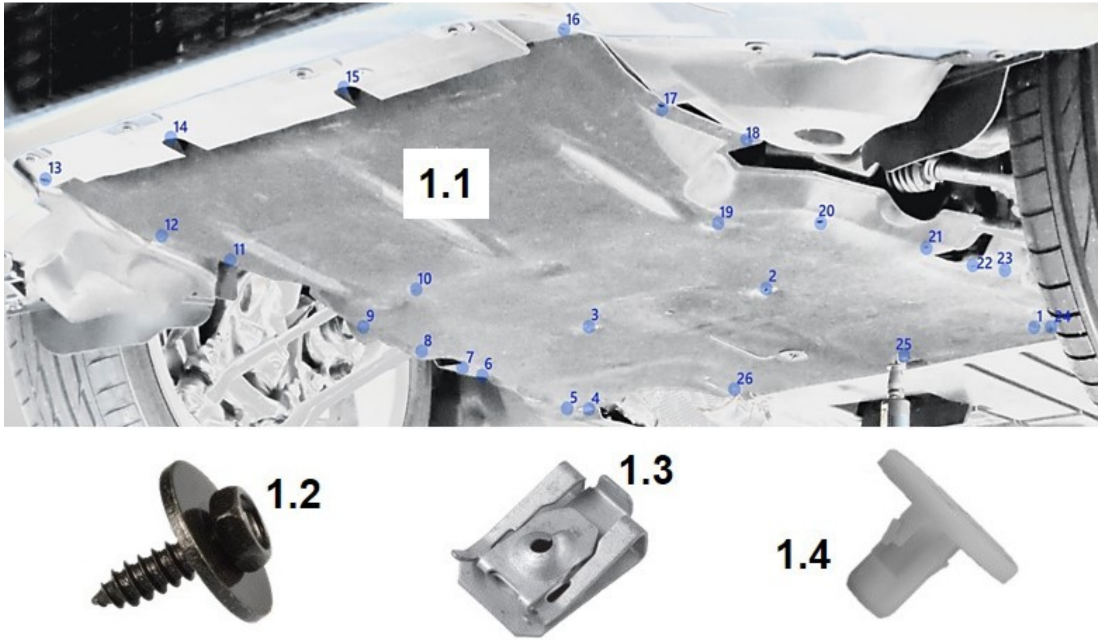
Thin panel with low rigidity that protects the engine and electronics

Assembly (simplified):
1. Install c-clips (1.3)
2. Install expanding nuts (1.4)
3. Install panel (1.1)
4. Fasten with screws (1.2)

Workers: 2 robots, 1 human

Workstation fits at most 2 of them
Some operations and tools are human/robot specific.

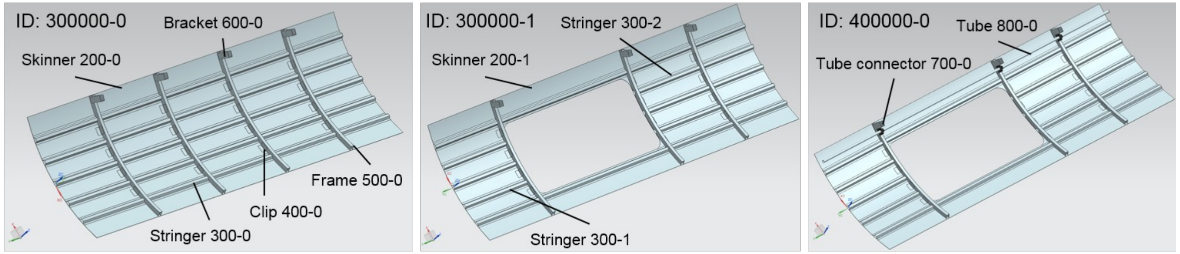
Case Study: Car Underbody Panel



Implementation effort	1	day
Tool input size	179	LoC
Classification time	0.3	min
Planning time	0.1	min
Schedule size	113	#actions

Case Study: Airplane Fuselage Shell

Image © Airbus

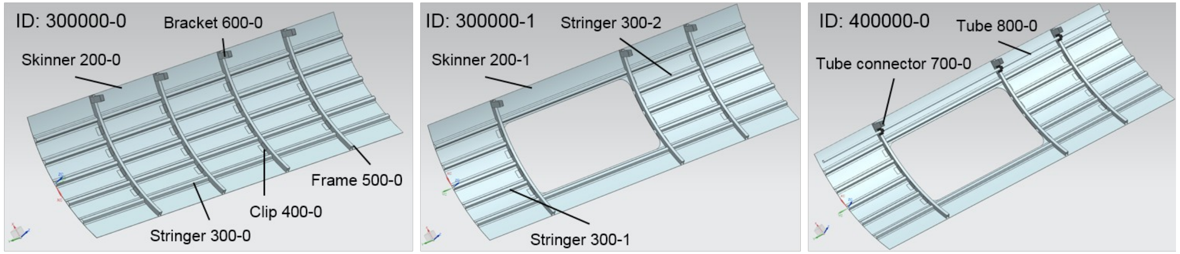


Fuselage shell element of section 13/14 of an Airbus A350

Assembly (extract):
1. The stringers and clips are glued to the shell
1.1 Apply glue
1.2 Join and fix elements
1.3 Activate the curing process with heat
1.4 Inspect
….

Case Study: Airplane Fuselage Shell

Image © Airbus



Implementation effort	2	days
Tool input size	501	LoC
Classification time	19	min
Planning time	15	min
Schedule size	170	#actions

Future Work

- Importing data from Product Life-cycle Management (PLM) systems (early prototype done)

  Connects to Siemens Teamcenter and generate Kotlin code.

- Graphical user interface
  - Ortwin: “Now I feel like a programmer” (mission accomplished ?!)
  - Programming not that common outside CS.
  - Can a scratch-like GUI work?

Take-Home Messages/Questions

- Software development and manufacturing have similar challenges (most of the “tools” contain computer)

  (Talk to them, there are interesting collaboration opportunities!)

- Automated reasoning technique (DL reasoner, SMT solver, etc.) are “too powerful” and, therefore, brittle. Especially in untrained hands. How can we fix that?
- Lot of work on DL and complexity. Maybe also applicable to software verification?