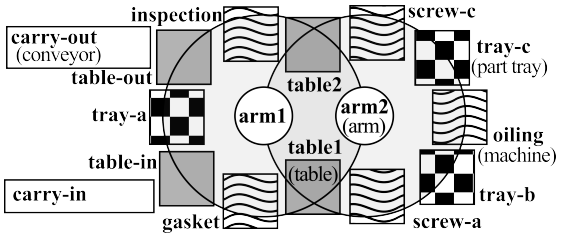
Fully Automated Cyclic Planning for Large-Scale Manufacturing Domains

Masataro Asai and Alex Fukunaga
The University of Tokyo

Mass-Manufacturing Domain and Loop Unrolling Strategy

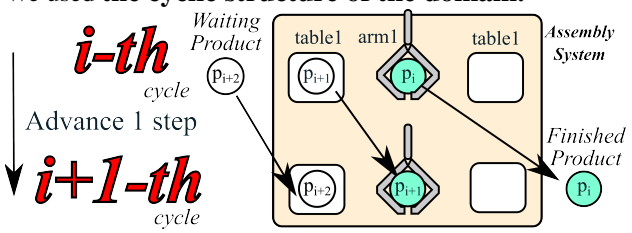


CELL-ASSEMBLY ≈ Gripper + Woodworking + Logistics (based on [Ochi et al, SPARK2013])

- Assemble and paint products while moving them with arms between tables or special purpose machines
- also a *temporal* problem (actions execute in parallel)

Large-scale problems in factory assembly require the manufacturing of **100's or 1000's of identical products.**

We used **the cyclic structure of the domain.**



Everything except indices of the products are the same after a cycle. Each loop path is identified by Steady State (SS) ≡ a state in the beginning of one cycle, indexed with i.

ex. S_i : (at p_{i+1} table), (at p_i arm), (painted p_i).

Build a large plan by unrolling the same loop.

Two challenges for fully automated cyclic plan synthesis

WHICH STATES are Steady States? : Problem 1

- Search all states from Init and test them → impractical.
- Labelling some objects (`table`, `machine`) and predicates (`at`) ? → Possible, but domain-dependent

Which Steady State yields the best cyclic plan? : Problem 2

- Brute force search → impractical
- ∵ testing each SS invokes a *standard STRIPS planner each time*

ACP : Finding Efficient Loops based on Domain Independent Analysis

1. Main contribution: Owner / Lock

A unit capacity resource ≈ a place.

use table1 ∈ Attatch @ table1.

A predicate being "a place or not"?

not trivial ∵ *syntactically they look the same*

Is this a place?	(human understanding)
(color ?p red)	no,maybe
(at ?p table1)	yes,maybe
(pred ?p X)	**hmm, no idea**

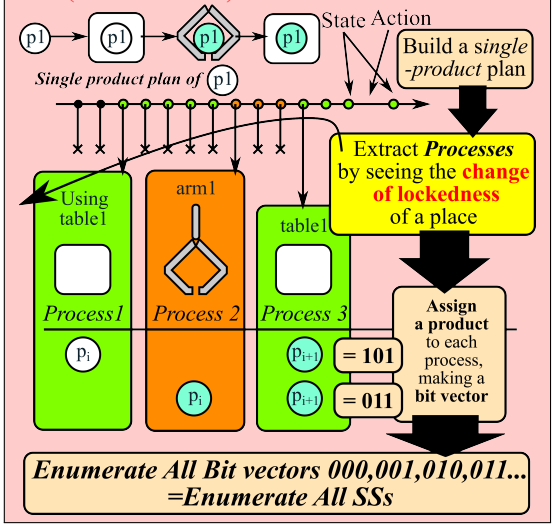
⇔ **Observation:** table1 may be "(used table1)".

Now we have a fully automated method: Predicate $o = (P\ X_1\ X_2 ...)$ is a place when there is a *lock* predicate l that satisfies:

1. If a occupies o, check if the place o is not in use, and acquire the lock l.
2. If a leaves o, release the lock l.

No assumptions about domain. → Applicable to any STRIPS input

2. Extract processes from a plan to enumerate SSs (Solves Prob.1).



3. Pruning SSs (Solves Prob.2) if they are:

	table1	arm1	table1
Infeasible	1	0	1
Deadlock	1	1	0
Duplicated*	1	0	0
	0	1	0

(* They are the same loop)

Highly effective - e.g., on CELL-ASSEMBLY problem instance, reduced # of candidates from 5×10^6 to 677.

Plan each cycle from S_i to S_{i+1}, get the least-makespan cyclic plan, and generate N instance plan by unrolling i, **with no additional computational cost!**

Experiments

- ACP vs 5 planners (FD/LMcut+postprocessing scheduler, FD/LAMA+postprocessing scheduler, yahsp, DAE, CPT) vs (best score of) 5 Simple Cyclic Planner (SCP).
- SCP: solve $K = \{1 \ldots 9\}$ products plan with above 5 planners and get the mean makespan per product. Some even failed to solve $K = 9$
- 5 CELL-ASSEMBLY, modified IPC Temporal Woodworking, unmodified IPC Barman.
- ACP solved **all instances**.
- Standard planners failed for $N \geq 16$.
- ACP was significantly better than SCP.
- Gap relative to lbound is reasonably good.
- **Automatically detected owner/lock structure in standard IPC problems** and generated cyclic solutions.

Problem	# of products	run-time	ACP makespan	makespan (per product)	SCP makespan (per product)	manual lbound	CPT(h2) lbound	gap (ACP / max. lbound)
	N	[sec]	c_{ACP}	c_{ACP}/N	c_{SCP}/K	l_m	l_{CPT}	
CELL-ASSEMBLY 1	4	1048	**331**	82.8	83 ($K = 2$) FD/LM$_{cut}$ (+ scheduler)	156	*176.3*	1.9
	16	1049	**1255**	78.4		*624*	460	2.0
	1024	1050	**78871**	**77.0**		*39936*	fail	2.0
CELL-ASSEMBLY 2	4	34	**246**	61.5	62.3 ($K = 3$) FD/LM$_{cut}$	168	*181.12*	1.4
	16	33	**978**	61.1		*672*	593	1.5
	1024	35	**62466**	**61.0**		*43008*	fail	1.5
CELL-ASSEMBLY 3	4	1893	**660**	165	171 ($K = 1$) FD/LAMA	176	*237*	2.8
	16	1953	**2352**	147		*704*	345	3.3
	1024	1973	**144480**	**141.1**		*45056*	fail	3.2
CELL-ASSEMBLY 4	4	1163	**318**	79.5	81.3 ($K = 3$) FD/LM$_{cut}$	112	*191*	1.7
	16	1162	**1074**	67.1		*448*	240	2.4
	1024	1165	**64578**	**63.1**		*28672*	fail	2.3
CELL-ASSEMBLY 5	4	1968	**804**	201	203 ($K = 1$) FD/LM$_{cut}$	172	*335*	2.4
	16	1856	**2508**	156.8		*688*	532	3.6
	1024	1894	**145644**	**142.2**		*44032*	fail	3.3
WW product : parts	4	11	**80**	20	17.2 ($K = 9$) FD/LAMA	60	*80*	1
	16	11	**260**	16.3		*240*	185	1.1
	1024	15	**15380**	**15.0**		*15360*	fail	1.0
Barman product : cocktail	4	331	35	8.8	**6.3** ($K = 4$) FD/LM$_{cut}$	4	*21*	1.7
	16	332	179	11.2		16	*26*	6.9
	1024	332	12275	12.0		*1024*	fail	12.0

Conclusions

Owner/lock method is applicable to **general STRIPS input**

Owner/lock exists in IPC domains

Solved large problems by simply unrolling the loop many times (negligible marginal cost)