Another Org-based Presentation

Masataro Asai

October 31, 2014

Fully automated cyclic planning for large-scale manufacturing domains.

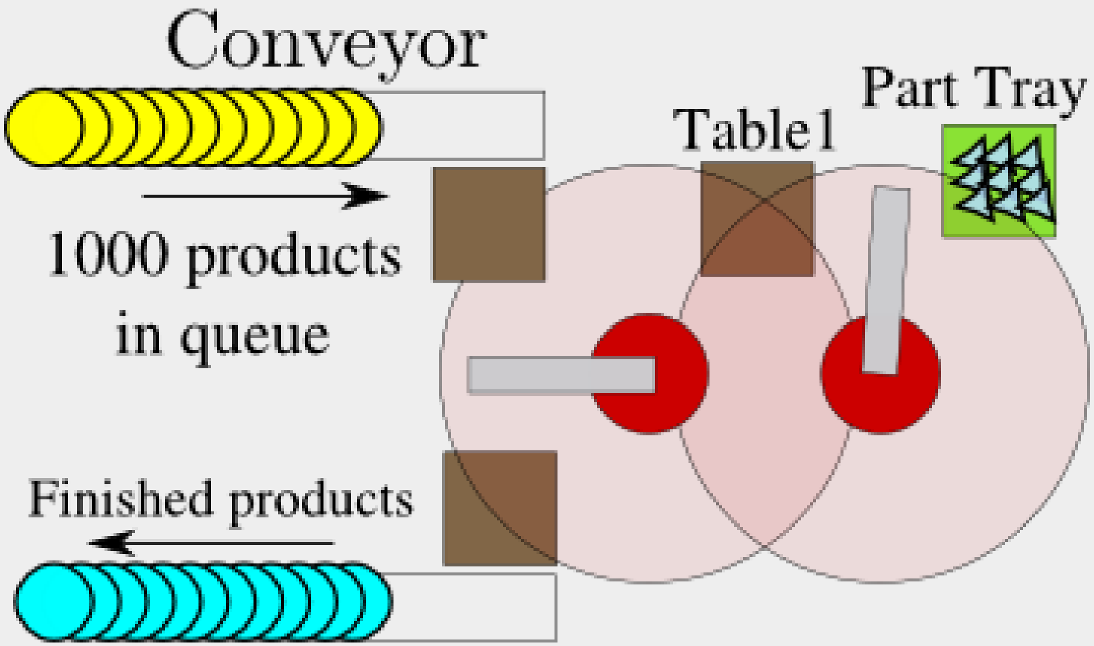
Masataro Asai Univ. tokyo

Alex Fukunaga (Assoc. prof, advisor)

1 Motion Planning in Cell Assembling System

- It's **NOT** about spatial search, all about **actions** !
- Require professional human resource w/o automated planner

2 CELL-ASSEMBLY

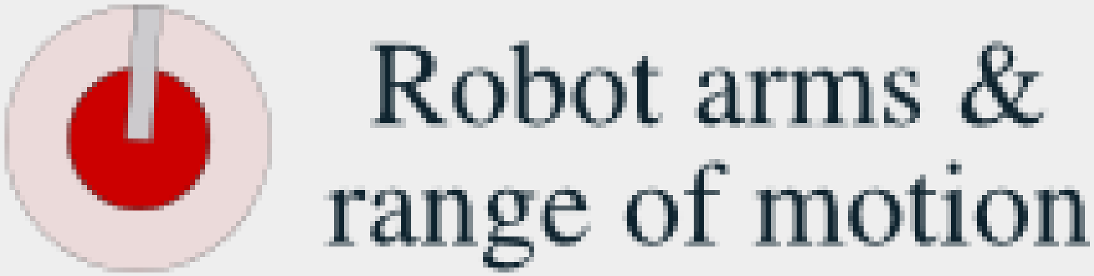


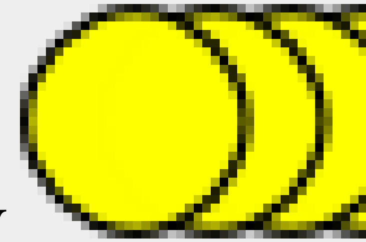
gripper + woodworking + logistics.

-  : many products, with

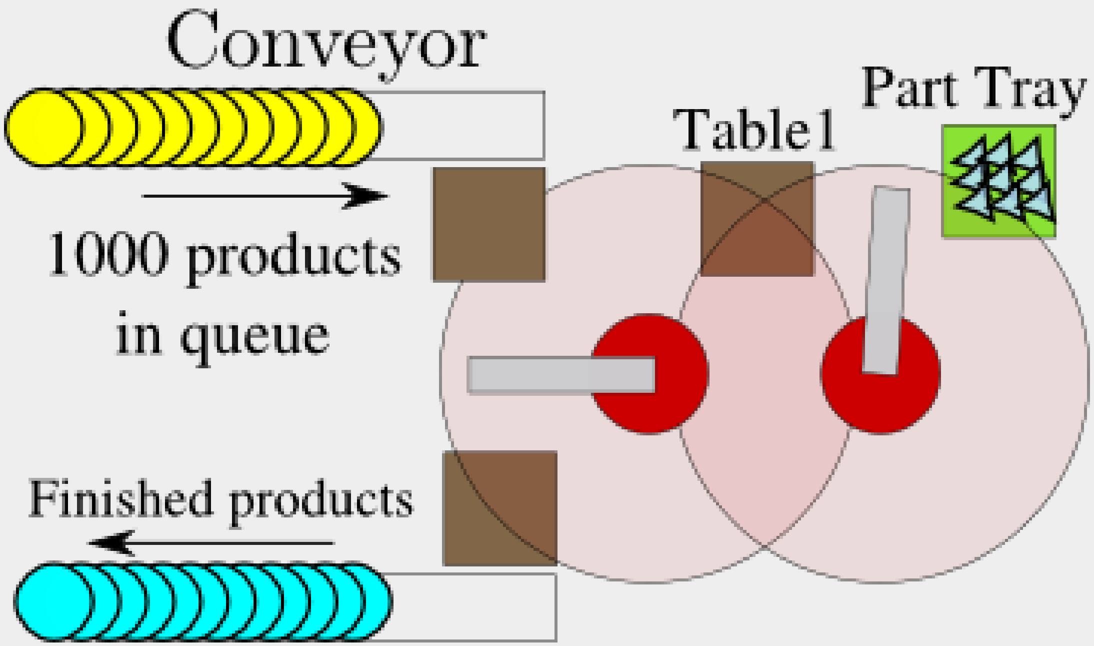
   : parts

- to complete each product, multiple operations – 

- 

moves to carry  without collision.

2.1 Planner's Task



Assembling recipes are provided in the problem

All products use the same recipe (Identical)

. . . (we tried to relax it in the KEPS paper!)

.

Primary Task : **optimizing the arm motion**

2.2 High degree of symmetry

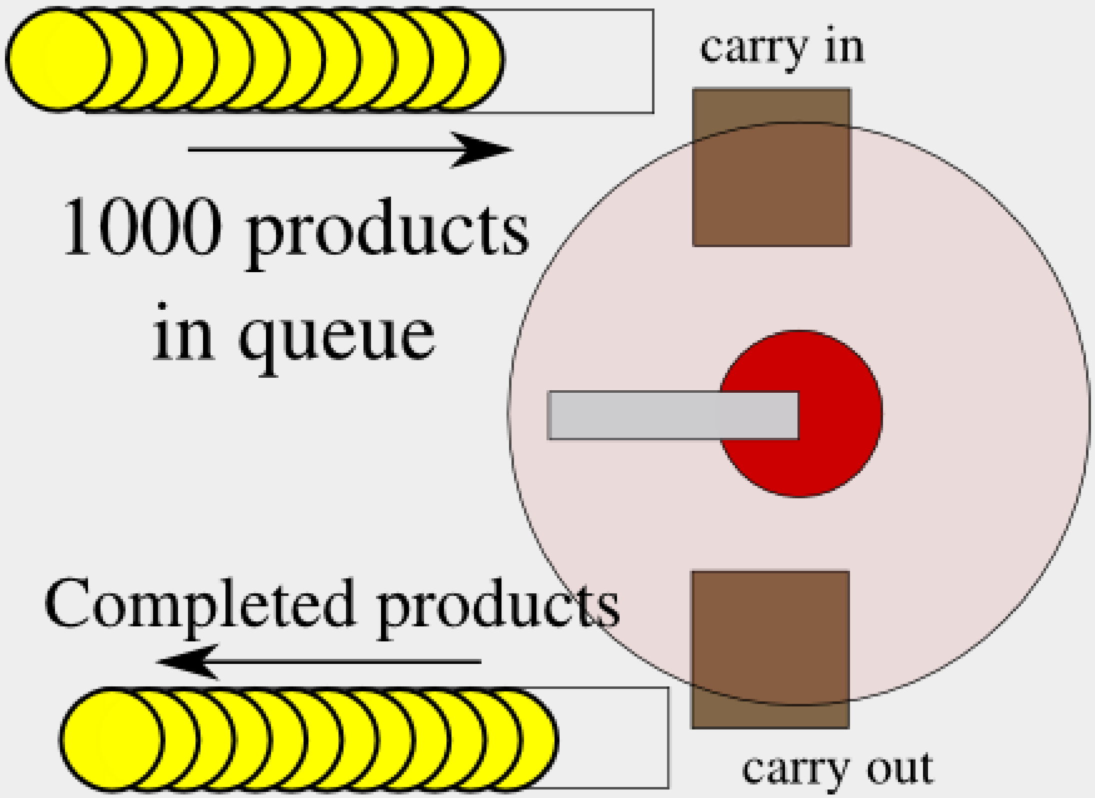


Figure 1: Simplest problem with many product

2.3 No Existing Planner Can Solve This Practical Problem!

- symmetry breaking
- <-> **ANOTHER METHOD**

		Standard Planner				
Problem	♯ of products	FD/LM$_{cut}$ + scheduler	FD/LAMA + scheduler	yahsp	DAE	CPT4
	N					
CELL-ASSEMBLY	4	fail	892	807	*774*	fail
2a	16	fail	fail	fail	fail	fail
(2 arms, 5 jobs)	64	fail	fail	fail	fail	fail
(τ = base)	256	fail	fail	fail	fail	fail
	1024	fail	fail	fail	fail	fail
CELL-ASSEMBLY	4	*249*	256	607	332	fail
	16	fail	fail	fail	fail	fail
2b (1a, 5j)	64	fail	fail	fail	fail	fail
	256	fail	fail	fail	fail	fail
	1024	fail	fail	fail	fail	fail

3 Summary of Contribution

Automated Framework to Form a Loop Structure .

First attempt: form and find the *best* cyclic plan

General domain/problem analysis method (owner/lock)

.

the basic method is applicable to other domains

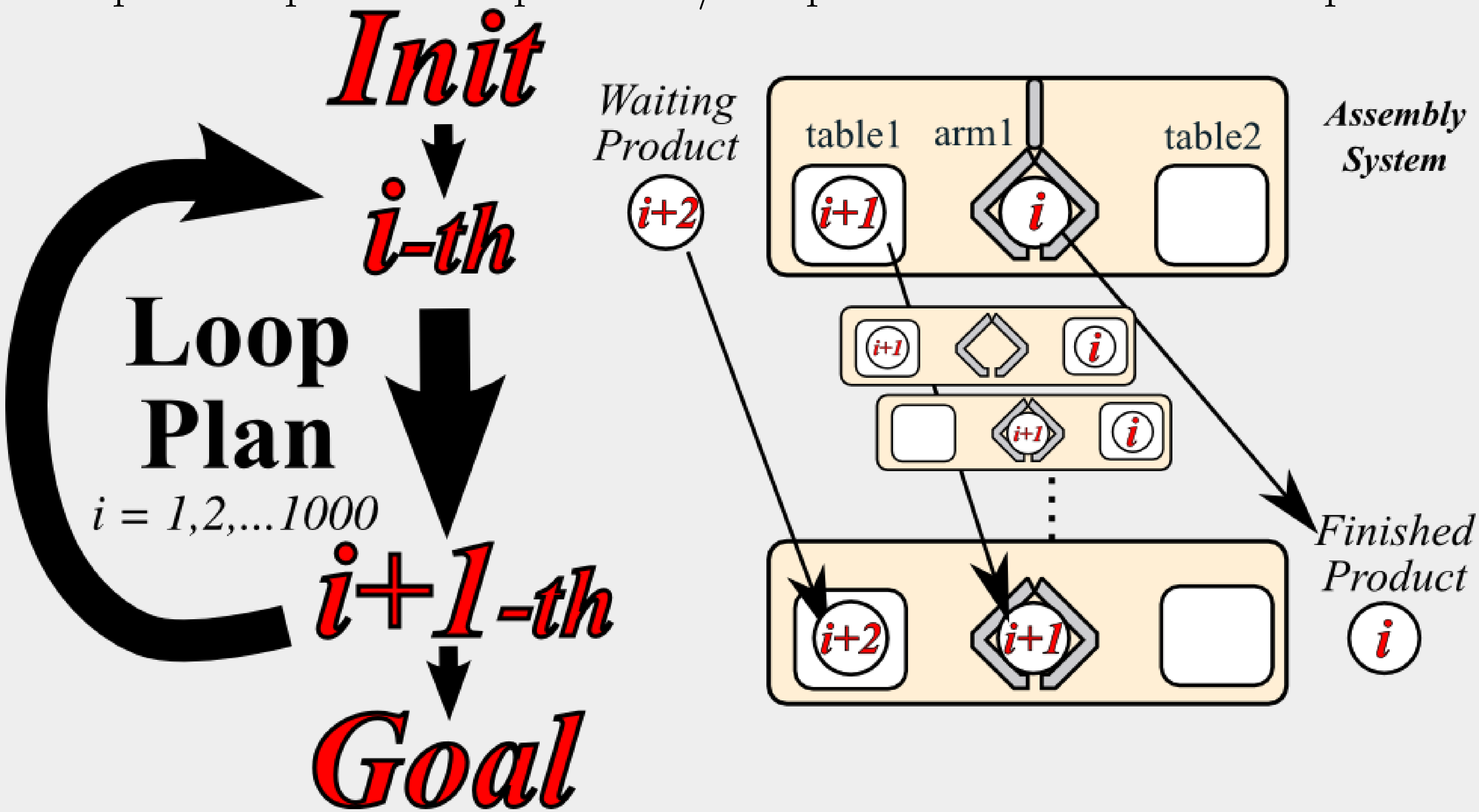
Solved EXTREMELY large PDDL instances (inc. IPC doma

.

This is *clearly beyond* state-of-the-art planners

4 Strategy: Cyclic Planning

One loop – completes one product / all products to the next steps



4.1 Loop unrolling <-> Loop *rolling*

length : 1000 x 3 = **3000**

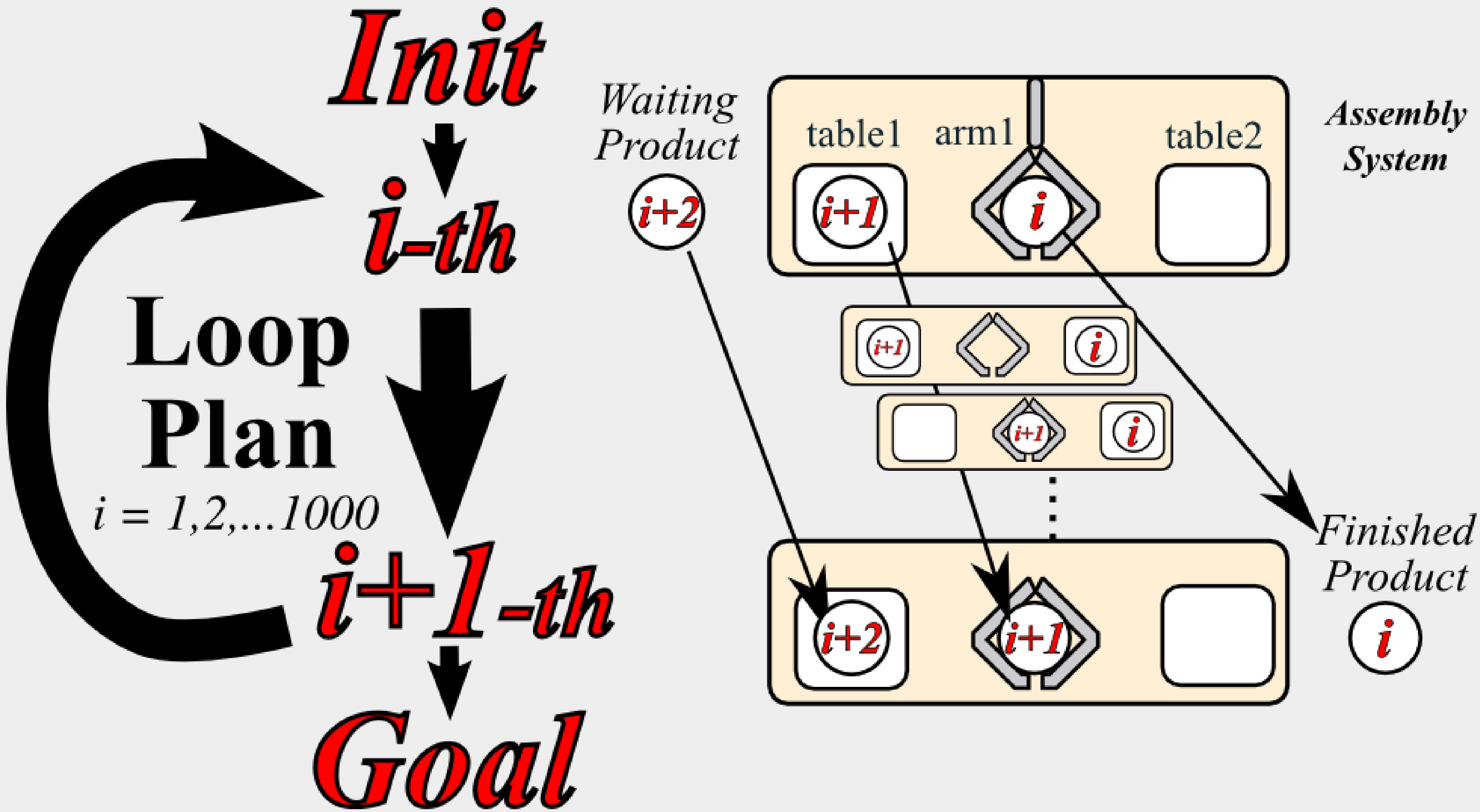
```
paint(A[0],table1)
weld(A[1],table2)
move(A[0],table1,table2)

paint(A[1],table1)
weld(A[2],table2)
move(A[1],table1,table2)
...
weld(A[1000],table2)
move(A[999],table1,table2)
```

length : 3

```
for i in [0..1000]
  paint(p[i],table1)
  weld(p[i+1],table2)
  move(p[i],table1,table2)
```

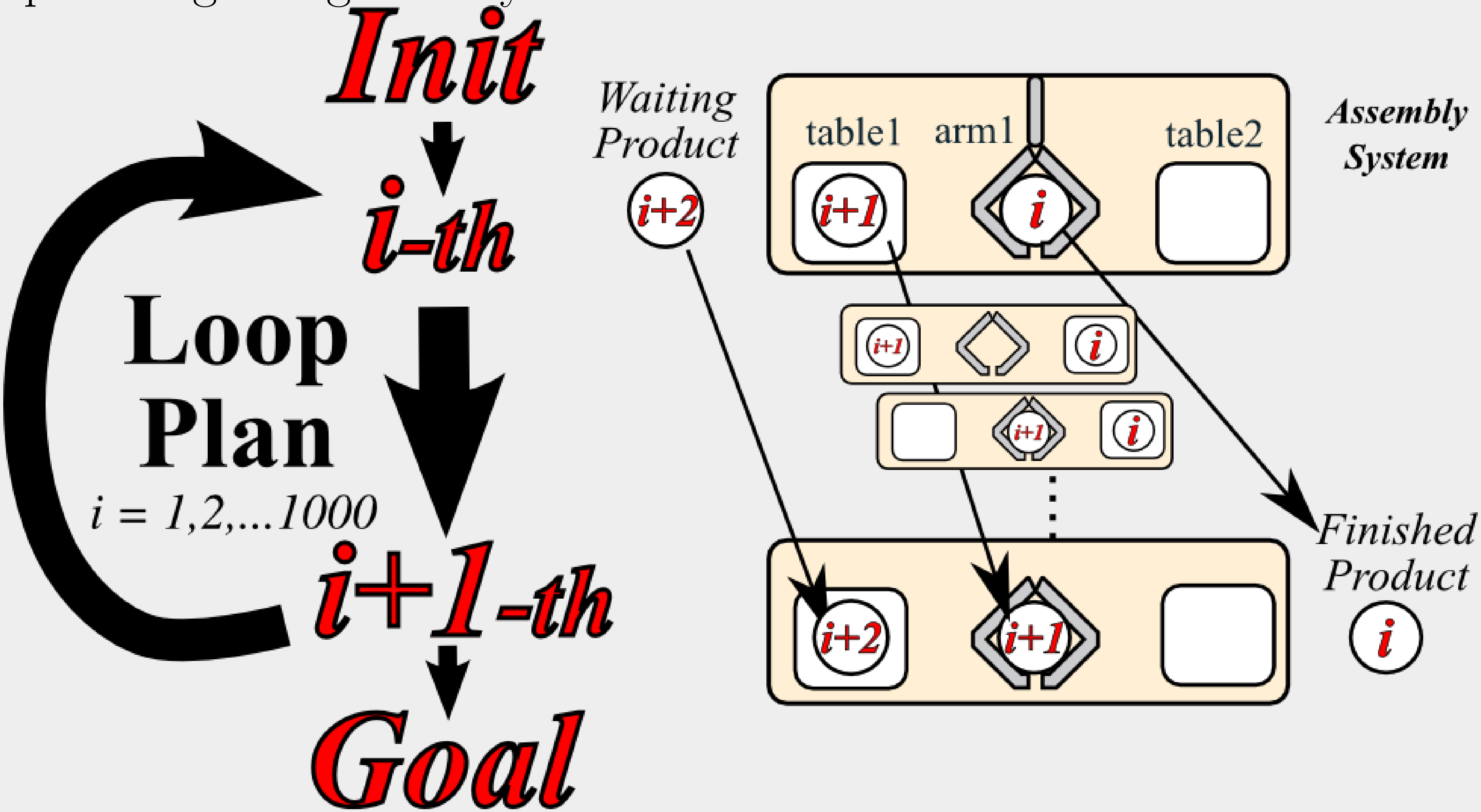
Planning is easier!

4.2 Interactions between subproblems



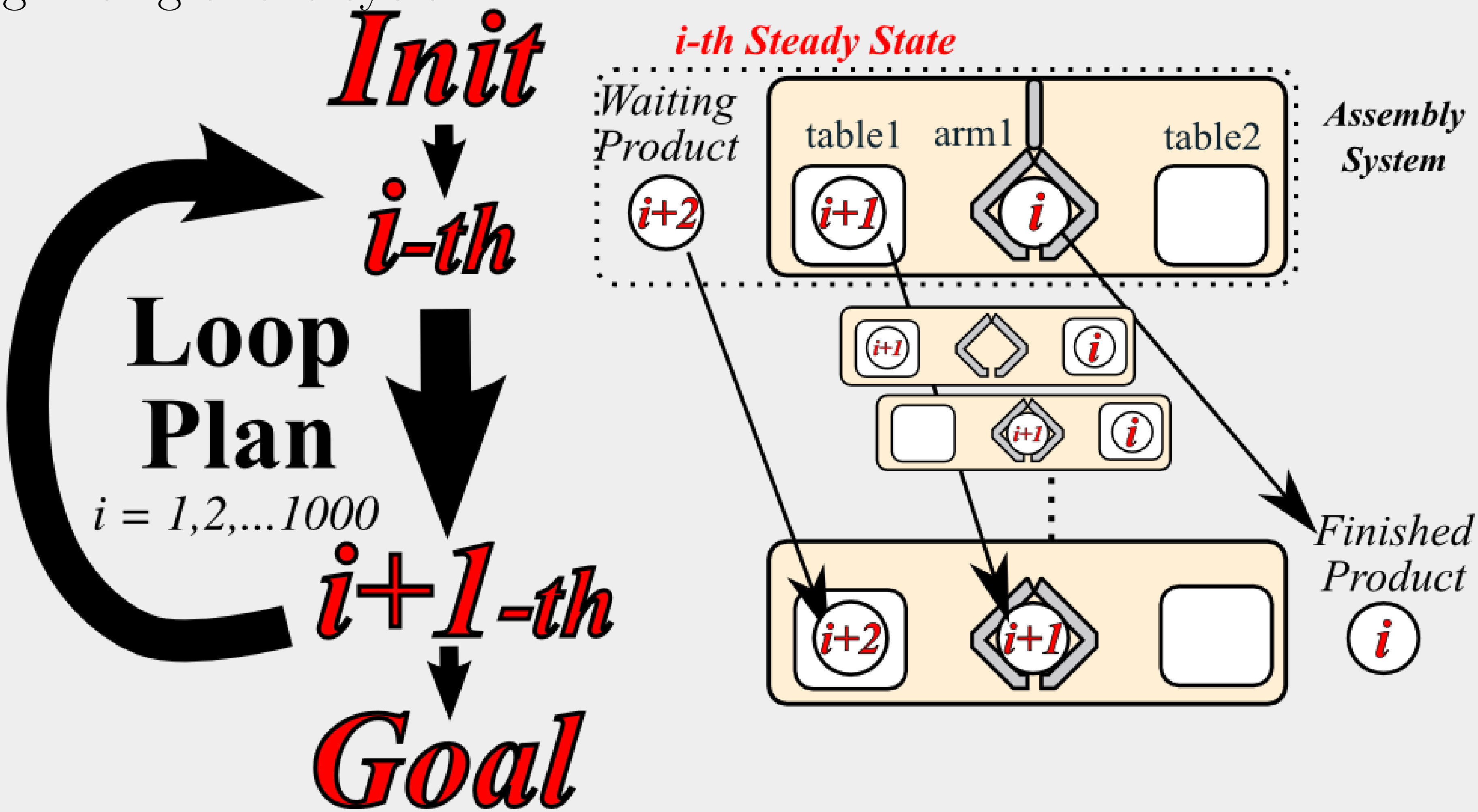
5 Loop : representation

Loop ⇔ Beginning of a Cycle

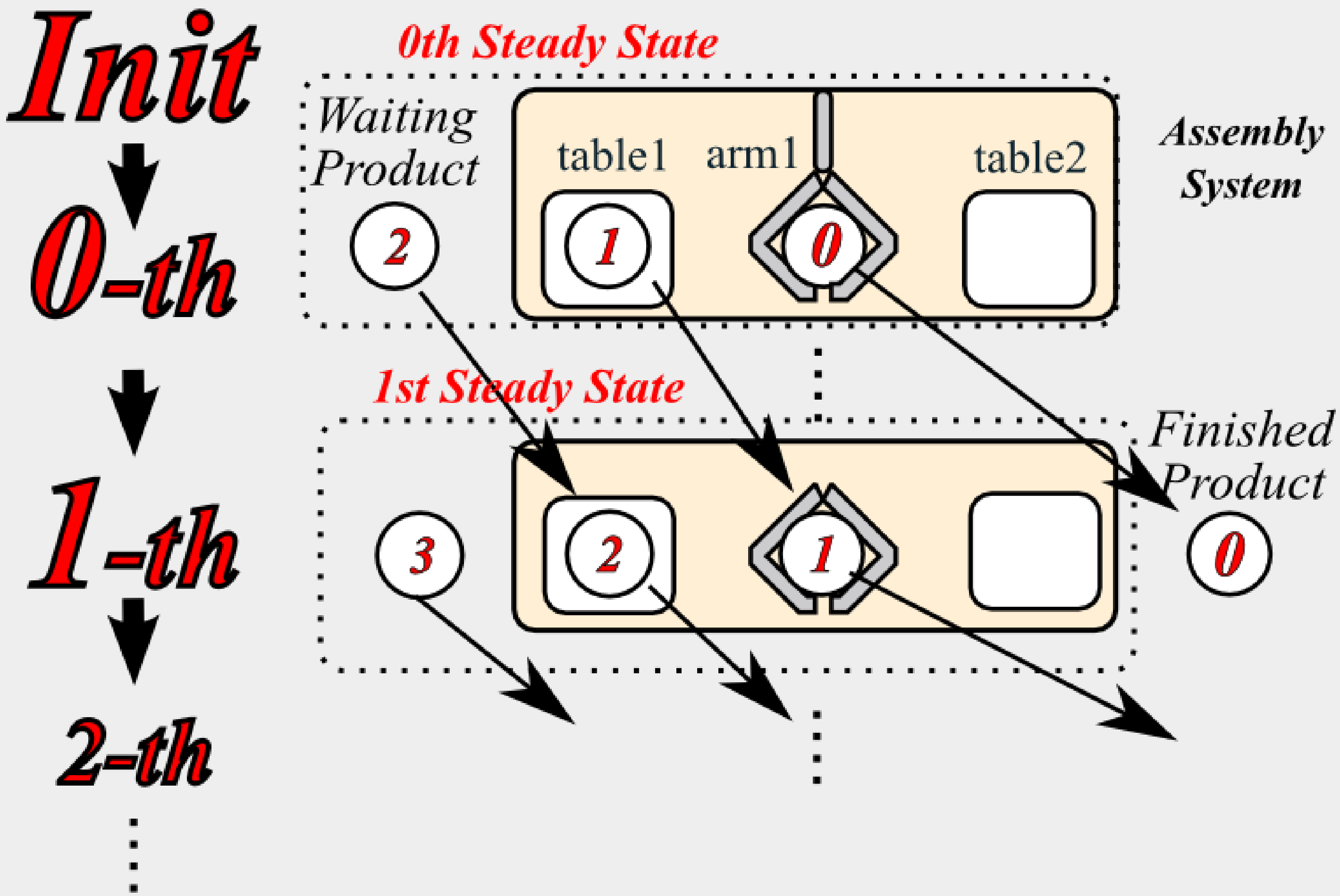


5.1 Steady State S_i

Beginneing of the cycle



5.2 Unrolling



6 Difficulty

It is not trivial to detect the information necessary for constructing an efficient loop.

6.1 Difficulty

WHICH STATES are the steady state? .

= which state can form a loop?

Checking ALL states from Init – impractical

Which SS yields a cost-efficient loop plan? .

Difficulty : (# of SSs) x (evaluation time for each SS)

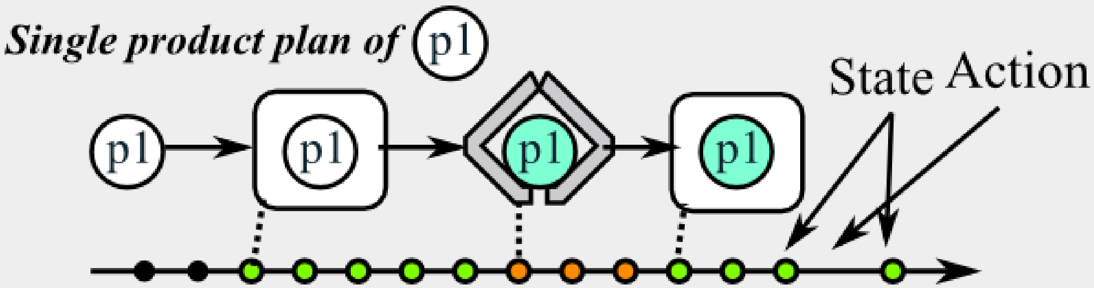
(# of SSs) – still exponentially large (e.g. 5x10^6)

(expensive evaluation) – loop plan : calls *FastDownward* each time

Even if we have a finite set of steady states,
<u>checking ALL steady states is again impractical!</u>

7 Process – to Enumerate SSs

Observation:



3 processes ($Attatch > \underline{Carry} > Attatch$)

There is at most **1** product in a process.

7.1 Process = Unit Capacity Resource

assemble @ table1 ≈ use table1 (resource)

- *Process* = {table, machine, painter} : *places*
- *Which predicate specifies a resource?* is not trivial

It's *difficult even for human!* w/o analysing PDDL
semantically indistinguishable – (X A Y)

	seems like..
(color X red)	not a place
(at X table)	a place
(P X Y)	??????

7.2 Detection mechanism

Question Is a predicate o= *(P X Y Z. . .)* a place?

Answer find a *lock* predicate l that satisfies **some condition** with o.

If it exists, it is a place.

Condition : for all action a,

1 . If a occupies a place, a should check if the place is not in use, and a should acquire the lock.

2 . If a leaves the place, release the lock.

7.3 What's the difference?

```
(:action put-on
 :parameters (?arm ?product ?table)
 :precondition (and (in ?arm ?product)
                    (not-in-use ?table))
 :effects (and (not (not-in-use ?table))
               (on ?product ?table)
               (clear ?arm)))
```

```
(:action paint
 :parameters (?product ?table ?color)
 :precondition (on ?product ?table)
 :effects (color ?table ?color))
```

Puts a product in an arm onto a table

Paint a product

7.4 What's the difference?

```
(:action put-on
 :parameters (?arm ?product ?table)
 :precondition (and (in ?arm ?product)
                    (not-in-use ?table))
 :effects (and (not (not-in-use ?table))
               (on ?product ?table)
               (clear ?arm)))
```

If an action **occupies a place,**
ensure the lock is free > **acquire the lock.**

```
(:action paint
 :parameters (?product ?table ?color)
 :precondition (on ?product ?table)
 :effects (color ?table ?color))
```

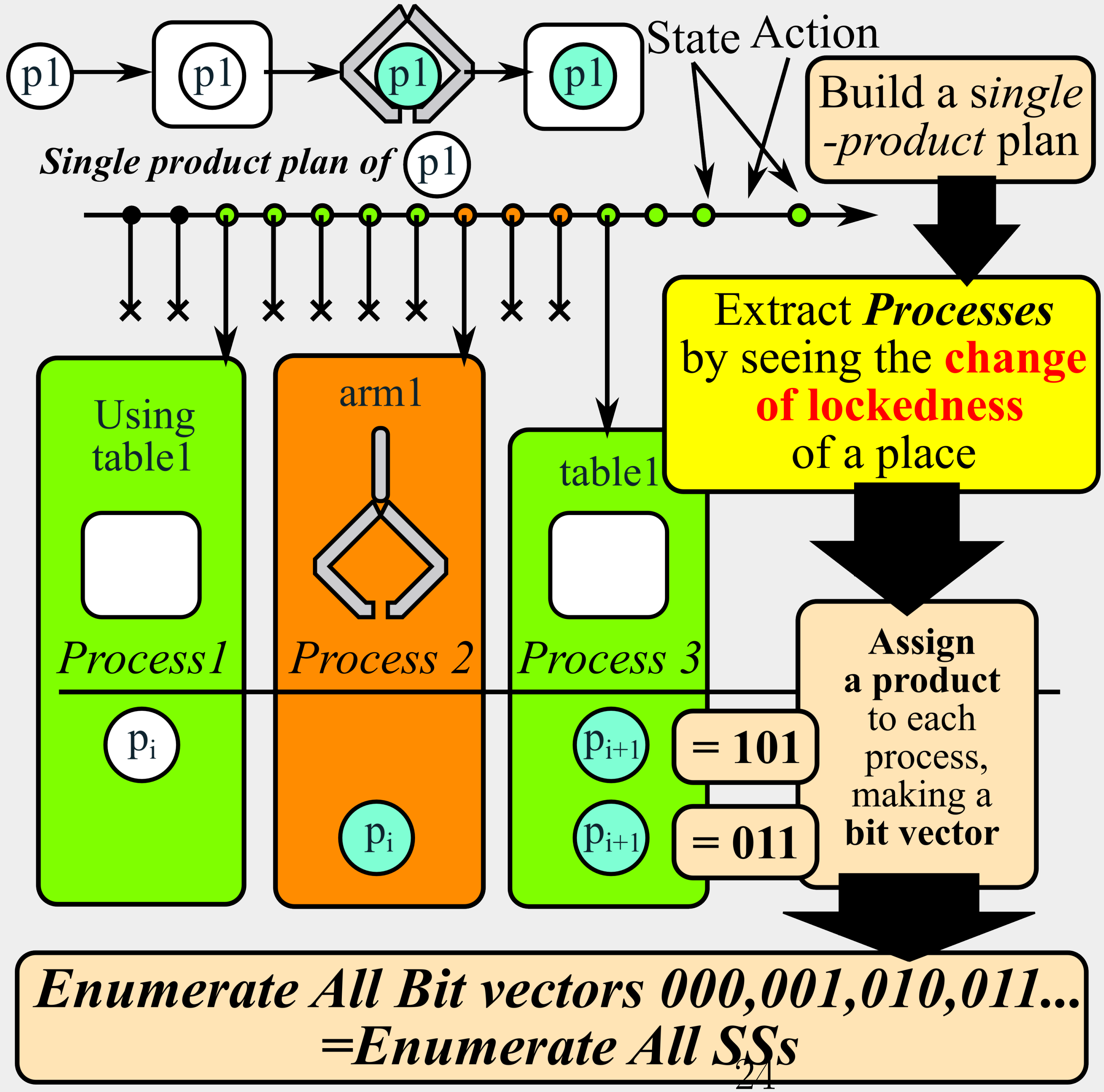
No such construct!

7.5 Detects ANY places possible

hiralious example, but…

(ሕበዋളპ ?ਆਛਟਊ ?嶺上開花 ?ฒฐญป) – non-place

7.6 ACP : Automated Cyclic Planner



7.7 Infeasibility/Deadlock Detection

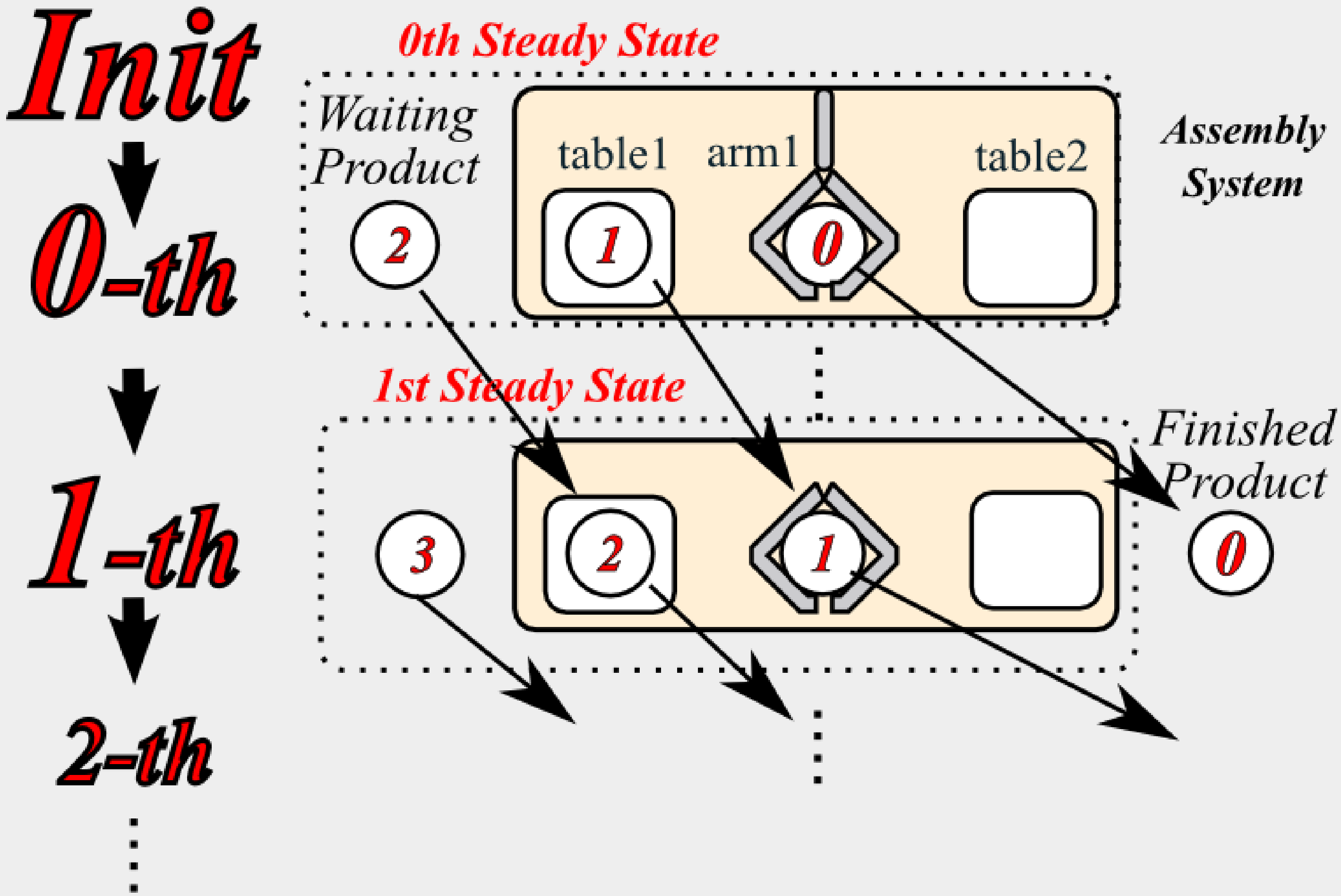
Using compact representation of steady state,

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

(* They are the same loop)

Reduced # : $5\text{x}10^6 \rightarrow 677$

8 Unrolling



Init
0-th
1-th
2-th
0th Steady State
Waiting
Product
table1
arm1
table2
Assembly
System
2
1
0
1st Steady State
Finished
Product
3
2
1
0

9 Experiments

- 5 CELL-ASSEMBLY problems (each with 4,16,64,256,1024 products)
- Woodworking (each with **same** 4,16,64,256,1024 parts)
- Barman (each with **same** 4,16,64,256,1024 cocktail)

9.1 ACP vs ...

- 5 temporal planners
  - FD/LAMA2011, FD/LMcut (+ min-slack scheduler)
  - yahsp2, DAEyahsp, CPT4
- best results of Simple Cyclic Planner (SCP)
  - 5 base planner x 9 configuration

9.2 Simple Cyclic Planner

Solve K = $\{1 \ldots 9\}$-products plan
with 5 planners (2FDs,yahsp,DAE,CPT)
get the average makespan per product (K-makespan/K)

9.3 All standard planners fails

		Standard Planner				
Problem	♯ of products	FD/LM$_{cut}$ + scheduler	FD/LAMA + scheduler	yahsp	DAE	CPT4
	N					
CELL-ASSEMBLY	4	fail	892	807	*774*	fail
2a	16	fail	fail	fail	fail	fail
(2 arms, 5 jobs)	64	fail	fail	fail	fail	fail
(τ = base)	256	fail	fail	fail	fail	fail
	1024	fail	fail	fail	fail	fail
CELL-ASSEMBLY	4	*249*	256	607	332	fail
	16	fail	fail	fail	fail	fail
2b (1a, 5j)	64	fail	fail	fail	fail	fail
	256	fail	fail	fail	fail	fail
	1024	fail	fail	fail	fail	fail

similar results on other domains

9.4 Average makespan compared to SCP / lower bound

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

9.5 Domain Independence

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

Too much assumptions & strong requirements?

10 Summary of Contribution

Automated Framework to Form a Loop Structure .

First attempt: form and find the *best* cyclic plan

General domain/problem analysis method .

the basic method is applicable to other domains

Solved EXTREMELY large PDDL instances .

beyond state-of-the-art planners

Solve Large IPC problem (and variants) correctly

10.1 Is it useful?

No one doesn't even try to solve that large problems!

Dirty attempt – lessons might be learned

Global lock/owner in STRIPS – it may find a way to use

10.2 So, what's next?

- Categorizing the objects into identical groups (KEPS paper)
  - several mixed-orders becomes available (100 x A / 200 x B)
  - (100 loops A) + (200 loops B)
  - (100 loops AB) + (100 loops B)
- Categorization -> Checks serial decomposability / not.
  - check if a resource is released or not
  - consider the "release" action of the resources as an abstract action
- **Unit** capacity -> **arbitrary** capacity
  - Detect **numbers** in a problem, automatically?
  - up-converting STRIPS to ADL (opposite to the common strategy)

Thanks for listening!