An Implementation of a Solver for Systems of Fixpoint Equations

Master's Thesis

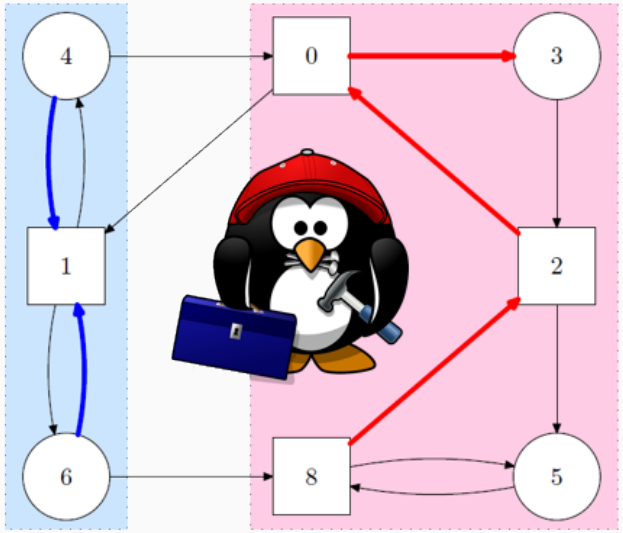
Paula Welzenbach

Advisors: Daniel Hausmann and Stefan Milius

21 July 2020

Chair for Theoretical Computer Science
Friedrich-Alexander Universität Erlangen-Nürnberg, Germany

Motivation



&

$$\mathsf{f}_j : \mathcal{P}(U)^{k+1} \to \mathcal{P}(U),\ U \subseteq \mathcal{P}(\mathbb{N})$$
$$f_0(V_0, \ldots, V_k) = \{V \in V_0 | \exists U' \in U.U' \subsetneq V\}$$
$$f_i(V_0, \ldots, V_k) = \{V \in V_{i-1} | \forall U' \in U.V \subseteq U'\}$$
$$f_k(V_0, \ldots, V_k) = V_0 \cup V_k$$
$$X_j =_{\mathtt{GFP}} f_j(X_0, \ldots, X_k) \text{ if } j \text{ even}$$
$$X_j =_{\mathtt{LFP}} f_j(X_0, \ldots, X_k) \text{ if } j \text{ odd}$$

Goal. Use generalized parity game algorithms to solve systems of fixpoint equations. (Image from Wikipedia)

Theoretical Background & Setting

Least & Greatest Fixpoint (Knaster & Tarski)

$h : \mathcal{P}(U) \to \mathcal{P}(U)$ monotone

- $\texttt{LFP}\ h = \bigcap \{V \in \mathcal{P}(U) | h(V) \subseteq V\}$
- $\texttt{GFP}\ h = \bigcup \{V \in \mathcal{P}(U) | V \subseteq h(V)\}$

Setting

$f_i : \mathcal{P}(U)^{k+1} \to \mathcal{P}(U)$, U finite and f_i monotone

Theoretical Background: Systems of Fixpoint Equations

Format

$X_i =_{\eta_i} f_i(X_0, \ldots, X_k)$,
where $\eta_i = \texttt{GFP}$ *if* i even, $\texttt{LFP}$ otherwise.

Semantics

$[\![X_i]\!]^\sigma = \eta_i X_i . f_i^\sigma$ with $\sigma : [k] \rightharpoonup \mathcal{P}(U)$,
where

$f_i^\sigma(A) = f_i([\![X_0]\!]^{\sigma'}, \ldots, [\![X_{i-1}]\!]^{\sigma'}, A, ev(\sigma', i+1), \ldots, ev(\sigma', k))$,

$\sigma' = \sigma[i \mapsto A]$,

$(\sigma[i \mapsto A])(j) = A$, *if* $j = i$, $\sigma(j)$ otherwise,

$ev(\sigma, i) = \sigma(i)$, if $i \in dom(\sigma)$, $[\![X_i]\!]^\sigma$ otherwise.

[Long *et al.*, 1994, Seidl, 1996]

Examples

- **Parity game:** $(U, E \subseteq U \times U, \Omega : U \to \mathbb{N})$, $U = V_\exists \dot\cup V_\forall$

$$
\begin{aligned}
f_0(V_0, \ldots, V_k) =& \{v \in V_\exists | E(v) \cap V_{\Omega(v)} \neq \emptyset\} \cup \\
& \{v \in V_\forall | E(v) \subseteq V_{\Omega(v)}\} \\
f_i(V_0, \ldots, V_k) =& V_{i-1} \qquad\qquad 0 < i \leq k
\end{aligned}
$$

- **Double powerset equation system:** $U \subseteq \mathcal{P}(\mathbb{N})$

$$
\begin{aligned}
f_0(V_0, \ldots, V_k) =& \{V \in V_0 | \exists U' \in U . U' \subsetneq V\} \\
f_i(V_0, \ldots, V_k) =& \{V \in V_{i-1} | \forall U' \in U . V \subseteq U'\} \qquad 0 < i < k \\
f_k(V_0, \ldots, V_k) =& V_0 \cup V_k
\end{aligned}
$$

Theoretical Background

$(\tilde{n}, \tilde{k}+1)$-Universal Graph [Czerwinski et al., 2018, Colcombet et al. 2018]

Even graph $G = (Z, L \subseteq Z \times [\tilde{k}] \times Z)$ with labels from $[\tilde{k}]$, such that for all even graphs G' with labels from $[\tilde{k}]$ and $|G'| \leq \tilde{n}$ there is a homomorphism from G' to G which transforms nodes but maintains labels.

Theoretical Background

$(\tilde{n}, \tilde{k}+1)$-Universal Graph [Czerwinski et al., 2018, Colcombet et al. 2018]

Even graph $G = (Z, L \subseteq Z \times [\tilde{k}] \times Z)$ with labels from $[\tilde{k}]$, such that for all even graphs G' with labels from $[\tilde{k}]$ and $|G'| \leq \tilde{n}$ there is a homomorphism from G' to G which transforms nodes but maintains labels.

Product Equation [Hausmann, Schröder, 2019]

$Y =_\nu g(Y)$ with $g : \mathcal{P}(U \times [\tilde{k}] \times Z) \to \mathcal{P}(U \times [\tilde{k}] \times Z)$, where

- $g(Y) = \{(v, p, q) \in U \times [\tilde{k}] \times Z | v \in f_p(X_0^q, \ldots, X_{\tilde{k}}^q)\}$
- $X_i^q = \{u \in U | \exists s \in L_i(q).(u, i, s) \in Y\}$
  $L_i(q) = \{q' | (q, i, q') \in L\}$

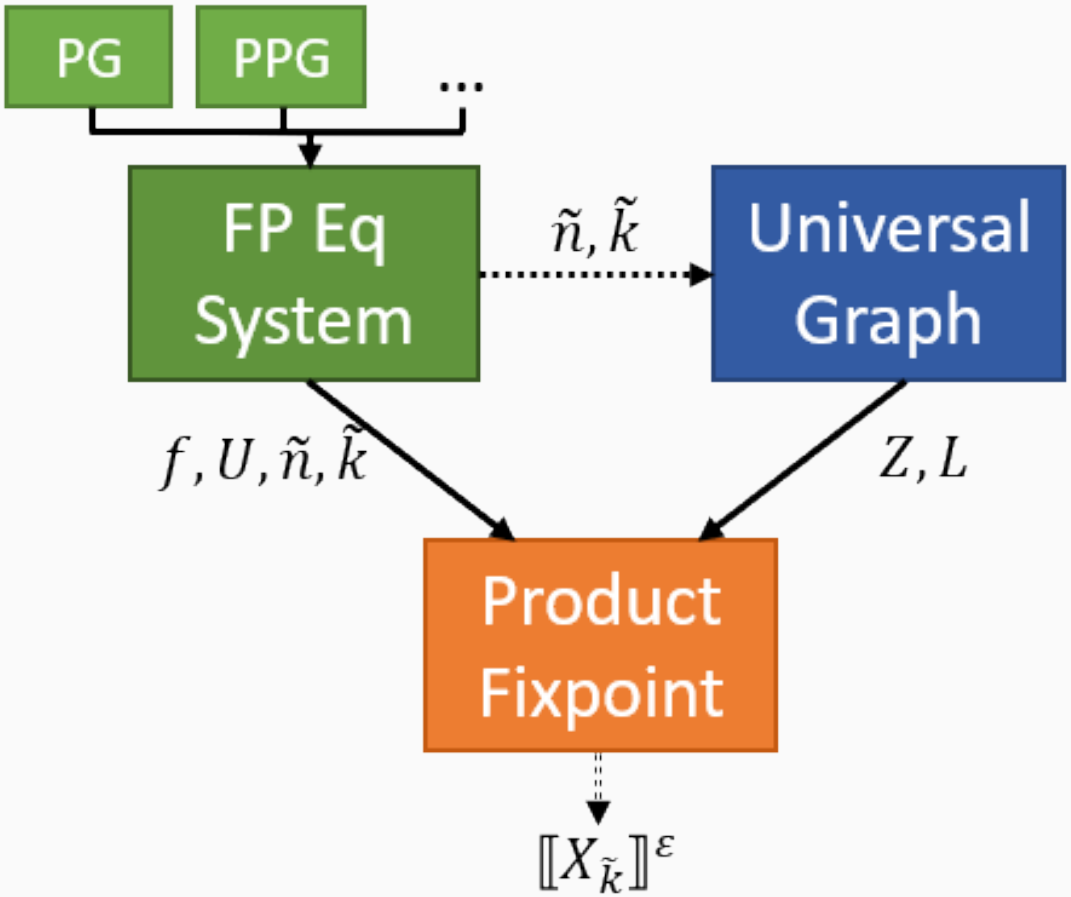
Theoretical Background

Theorem [Hausmann, Schröder, 2019]

For $0 \leq i \leq \tilde{k}$: $u \in [\![X_i]\!]_f \Leftrightarrow (u, i) \in \tau[[\![Y]\!]]$, where $\tau(v, p, q) = (v, p)$

$\rightsquigarrow$ Use universal graph (UG) and product equation to solve fixpoint equation systems.

Theoretical Background: Universal Graph & Product Fixpoint



$(n(k+1), k+1)$-Universal Graphs: Exponential & Quasi-polynomial

Exponential Universal Graph [Colcombet et al., 2018]

- Nodes: $(m_{k+1}, m_{k-1}, \ldots, m_1)$, $m_i \in [n(k+1)]$ (k even)
  $\rightsquigarrow$ "Timeouts": How often is a priority still allowed to occur?
- Size: $(n(k+1))^{k/2+1}$

$(n(k+1), k+1)$-Universal Graphs: Exponential & Quasi-polynomial

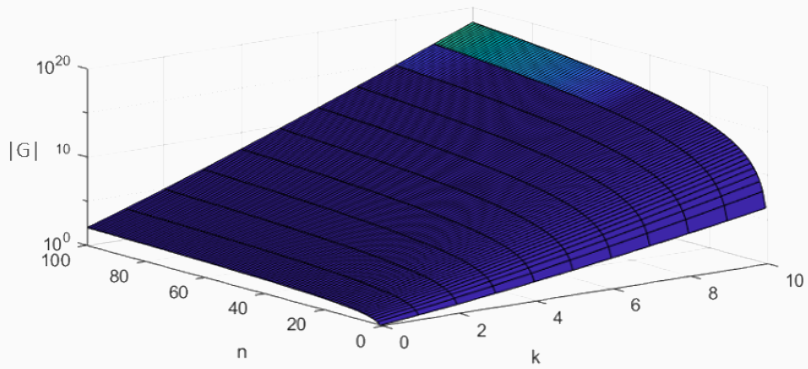
Exponential Universal Graph [Colcombet et al., 2018]

- Nodes: $(m_{k+1}, m_{k-1}, \ldots, m_1)$, $m_i \in [n(k+1)]$ (k even)
  $\rightsquigarrow$ "Timeouts": How often is a priority still allowed to occur?
- Size: $(n(k+1))^{k/2+1}$

Quasi-polynomial Universal Graph [Calude et al., 2017]

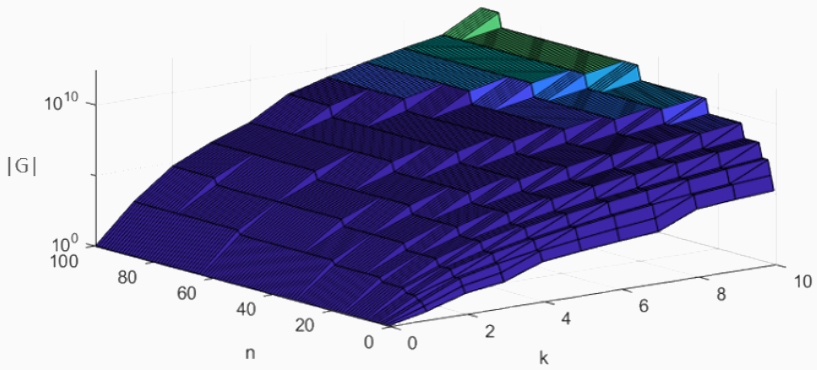
- Nodes: $(b_{\lceil log(n(k+1))\rceil}, .., b_0)$, $b_i \in [k]$
  $\rightsquigarrow$ "History": How often has a priority occured?
- Size: $(k+1)^{\lceil log(n(k+1))\rceil+1}$

$(n(k+1), k+1)$-Universal Graphs: Exponential & Quasi-polynomial



Exponential Universal Graph:

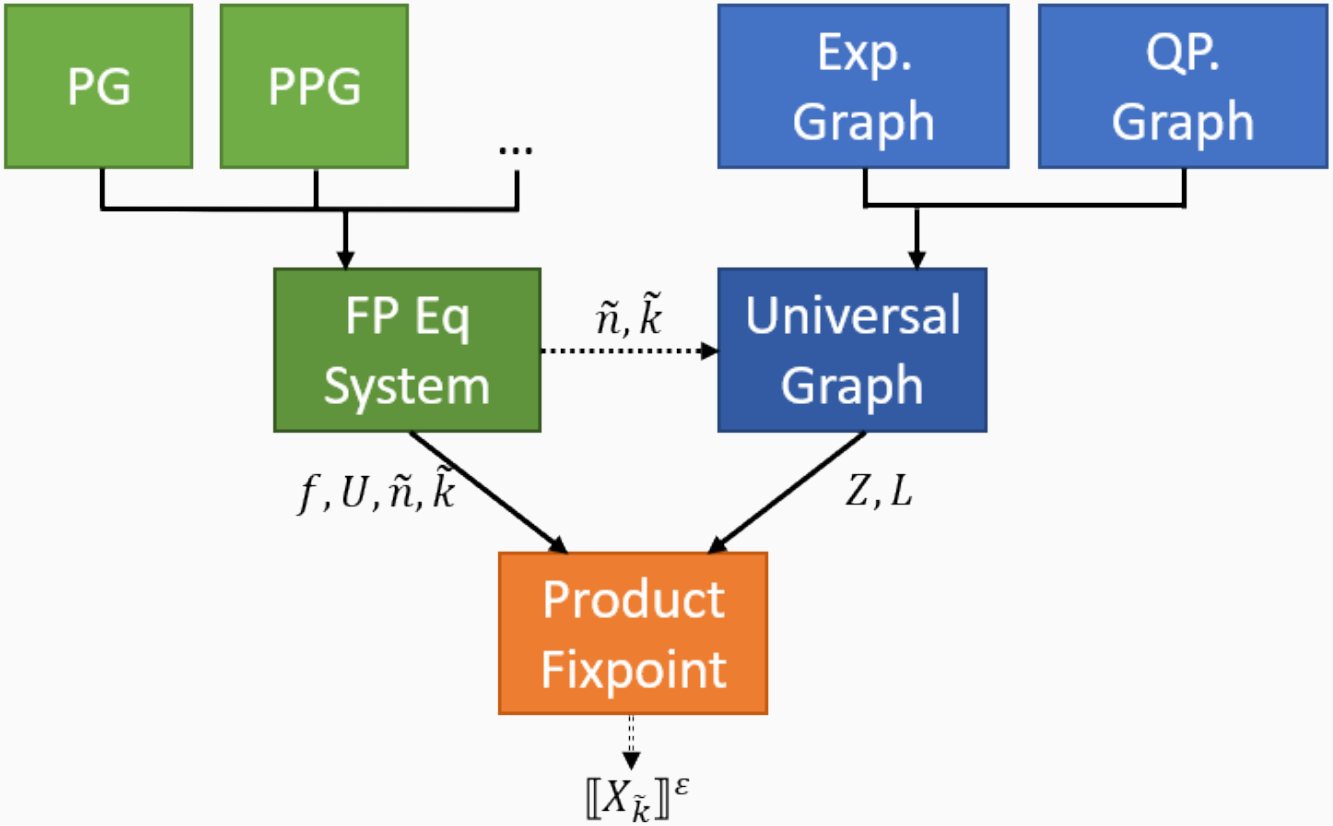
$|G| = (n(k+1))^{k/2+1}$



Quasi-polynomial Universal Graph:

$|G| = (k+1)^{\lceil log(n(k+1)) \rceil + 1}$

Theoretical Background: Universal Graphs & Product Fixpoint



Product Fixpoint: Three Variants

"Negative" Variant $\rightsquigarrow$ $Y =_\nu g(Y)$

- $g(Y) = \{(v, p, q) \in U \times [\tilde{k}] \times Z | v \in f_p(X_0^q, \ldots, X_{\tilde{k}}^q)\}$
- $X_i^q = \{u \in U | \exists s \in L_i(q).(u, i, s) \in Y\}$
- $\exists q.(v, \tilde{k}, q) \in Y$

Product Fixpoint: Three Variants

"Negative" Variant $\rightsquigarrow Y =_\nu g(Y)$

- $g(Y) = \{(v, p, q) \in U \times [\tilde{k}] \times Z | v \in f_p(X_0^q, \ldots, X_{\tilde{k}}^q)\}$
- $X_i^q = \{u \in U | \exists s \in L_i(q).(u, i, s) \in Y\}$
- $\exists q.(v, \tilde{k}, q) \in Y$

"Dual Negative" Variant $\rightsquigarrow Y =_\mu \overline{g}(Y)$

- Using the principle of duality: $\mu g = \overline{\nu \overline{g}}$ with $\overline{g}(Y) := \overline{g(\overline{Y})}$
- $\overline{g}(Y) = \{(v, p, q) \in U \times [\tilde{k}] \times Z | v \notin f_p(\overline{X_0}^q, \ldots, \overline{X_{\tilde{k}}}^q)\}$
- $\overline{X_i}^q = \{u \in U | \exists s \in L_i(q).(u, i, s) \notin Y\}$
- $\exists q.(v, \tilde{k}, q) \notin Y$

Product Fixpoint: Three Variants

"Negative" Variant $\rightsquigarrow Y =_\nu g(Y)$

- $g(Y) = \{(v, p, q) \in U \times [\tilde{k}] \times Z | v \in f_p(X_0^q, \ldots, X_{\tilde{k}}^q)\}$
- $X_i^q = \{u \in U | \exists s \in L_i(q).(u, i, s) \in Y\}$
- $\exists q.(v, \tilde{k}, q) \in Y$

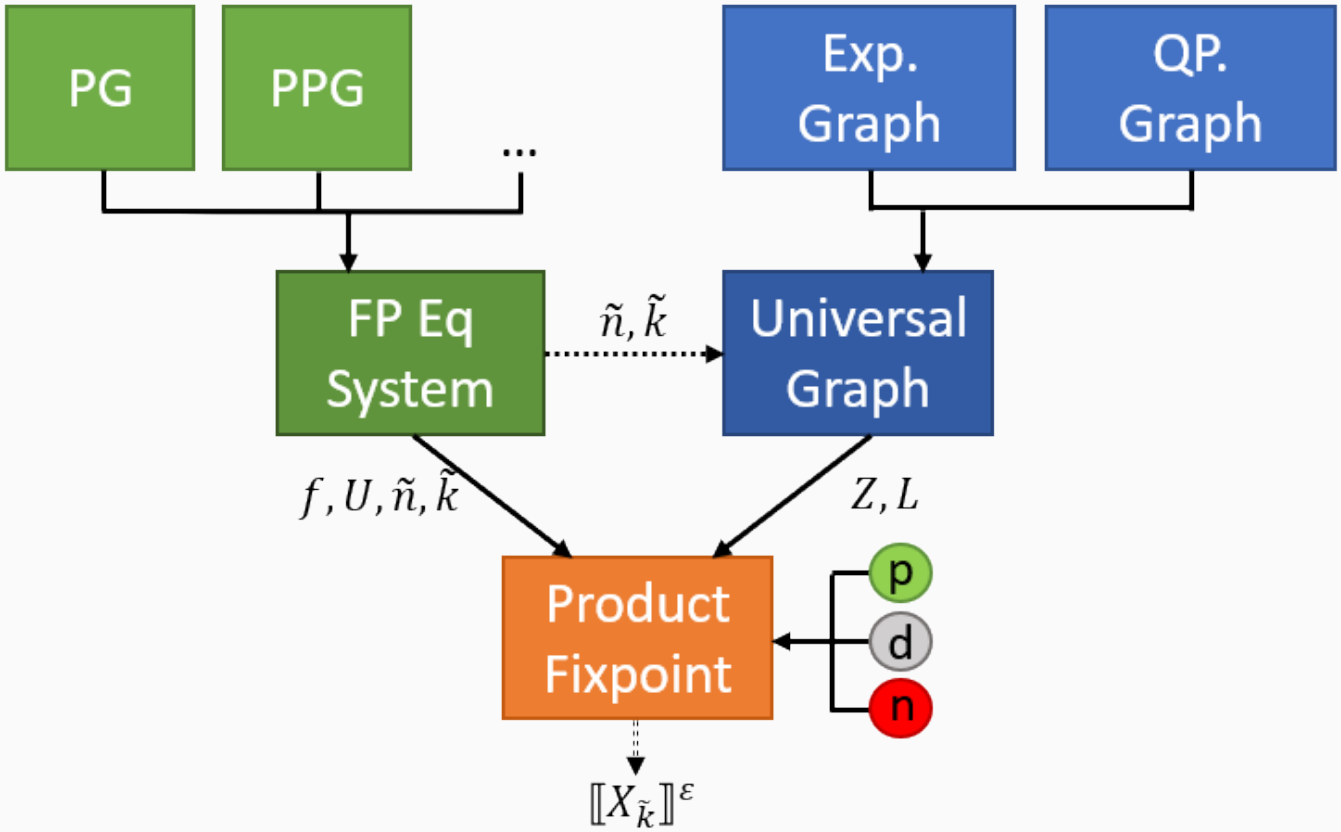
"Dual Negative" Variant $\rightsquigarrow Y =_\mu \overline{g}(Y)$

- Using the principle of duality: $\mu g = \overline{\nu \overline{g}}$ with $\overline{g}(Y) := \overline{g(\overline{Y})}$
- $\overline{g}(Y) = \{(v, p, q) \in U \times [\tilde{k}] \times Z | v \notin f_p(\overline{X_0}^q, \ldots, \overline{X_{\tilde{k}}}^q)\}$
- $\overline{X_i}^q = \{u \in U | \exists s \in L_i(q).(u, i, s) \notin Y\}$
- $\exists q.(v, \tilde{k}, q) \notin Y$

"Positive" Variant $\rightsquigarrow Y =_\mu g(Y)$

- $g(Y) = \{(v, p, q) \in U \times [\tilde{k}] \times Z | v \in f_p(X_1^q, \ldots, X_{\tilde{k}+1}^q)\}$
- $X_i^q = \{u \in U | \forall s \in L_i(q).(u, i-1, s) \in Y\}$
- $(v, \tilde{k}, max) \in Y$

Theoretical Background: Universal Graphs & Product Fixpoint



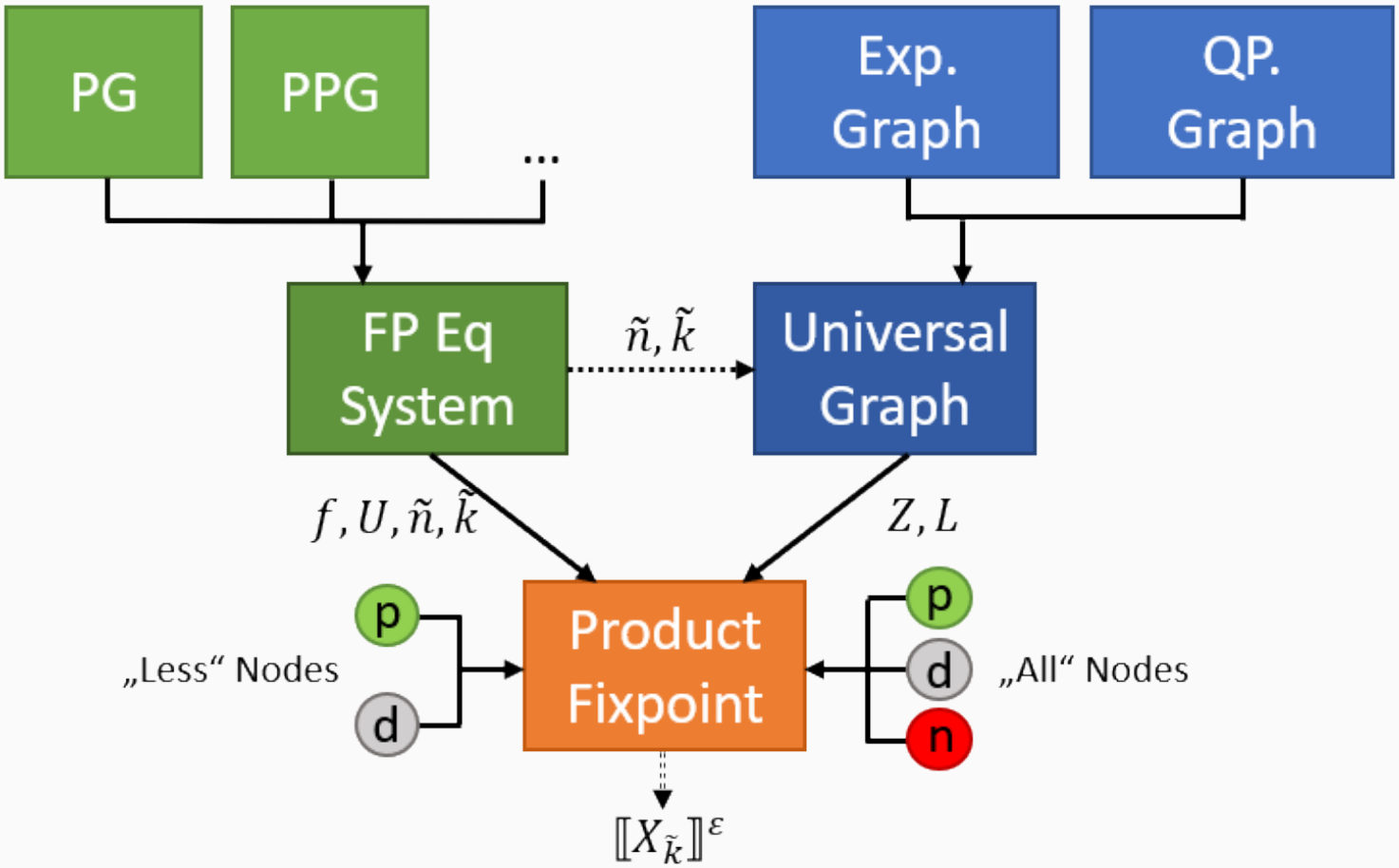
Two Algorithms for the Product Fixpoint: "All"/"Less" Nodes

Is it necessary to compute the whole universal graph in advance?

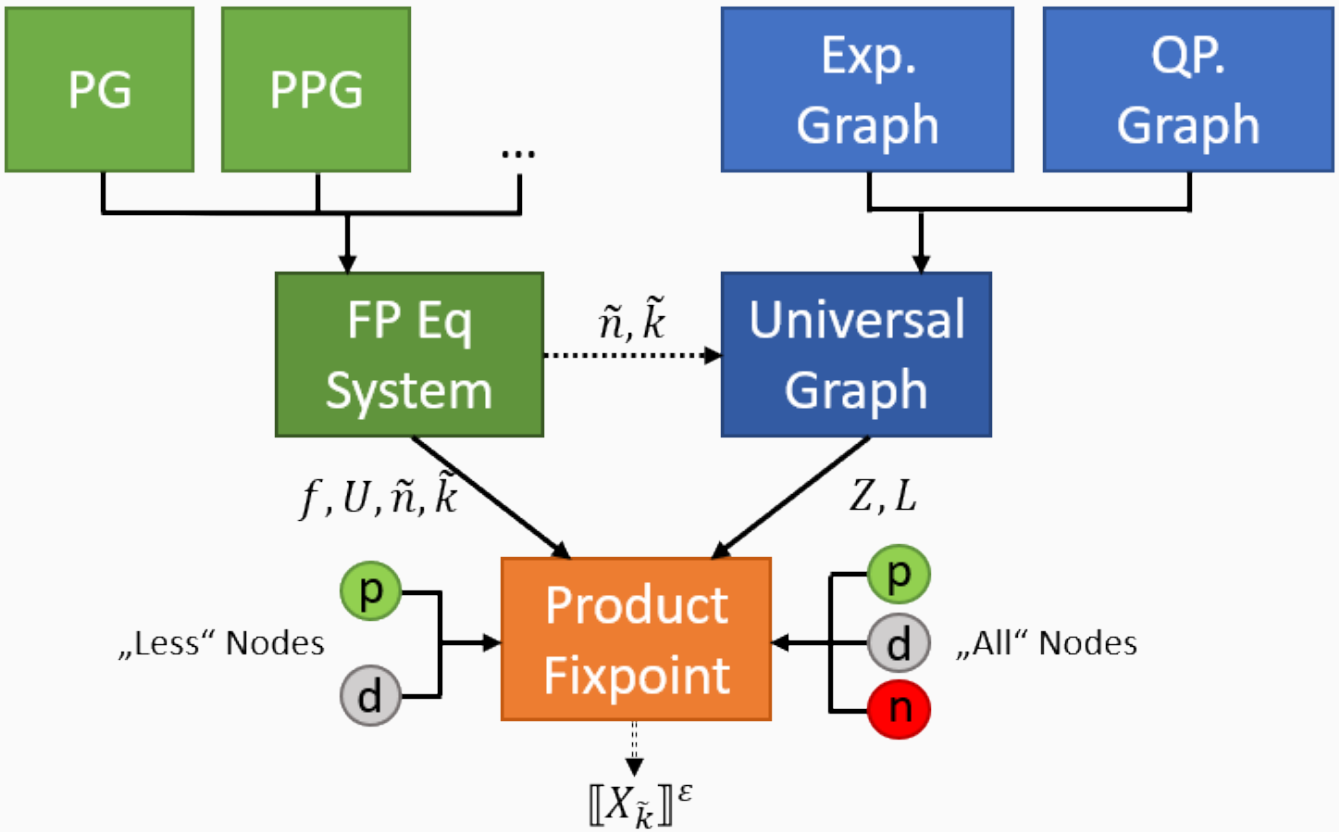
- Negative variant: Yes.
- Positive/Dual Negative variant: No!
  $\rightsquigarrow$ Start with leaves and investigate their predecessors
  - Start with $Y := \{(v, p, q) | v \in U, p \in [\tilde{k}] \text{ odd}, L_p(q) = \emptyset\}$
  - p-leaves for the exp. UG: $m_p = 1$
  - p-leaves for the qp. UG: only odd entries

  $\rightsquigarrow$ Less nodes in the graph, faster execution!

Universal Graphs & Product Fixpoint: Full Overview



Implementation: Architecture



Strict encapsulation, "Strategy" design pattern $\rightsquigarrow$ simple extensibility

Implementation: Data Structures

```
(* Universal Graph *)
type graph = {
  edges: node NodeLabelHT.t;                    (* ((n,p),m) *)
  predecessors: (unit NodeHT.t) NodeHT.t; (* (m,N) *)
  leaves: unit NodeLabelHT.t;                  (* ((n,p),()) *)
  nodes: unit NodeHT.t;                           (* (n,()) *)
  n: int;
  k: int
}

(* Exponential/Quasi-polynomial Graph *)
type node = int array

(* Task *)
module AbstractUSet = Set.S with type t=utype
```

Implementation: "All Nodes"

Negative Variant

procedure FPApproximation($f, U, \tilde{n}, \tilde{k}$)
  $G \leftarrow$ negativeGraph($\tilde{n}, \tilde{k}$)
  $Y \leftarrow U \times [\tilde{k}] \times G$.nodes
  $Y' \leftarrow \emptyset$
  firstRun $\leftarrow$ true
  while $Y' \neq Y$ **do**
    if not firstRun **then**
      $Y \leftarrow Y'$
    firstRun $\leftarrow$ false
    for q in G.nodes **do**
      for i from 0 to $\tilde{k}$ **do**
        $X_i^q \leftarrow \{u \in U | \exists s \in L_i(q).(u, i, s) \in Y\}$
      temp $\leftarrow \emptyset$
      for i from 0 to $\tilde{k}$ **do**
        res $\leftarrow f_i(X_0^q, \ldots, X_{\tilde{k}}^q)$
        temp $\leftarrow$ temp $\cup \{(v, i, q) | v \in \text{res}\}$
      $Y' \leftarrow$ temp
  return Y

Implementation: "Less Nodes"

Positive Variant

procedure FPApproximation($f, U, \tilde{n}, \tilde{k}$)
  $G \leftarrow$ emptyPositiveGraph($\tilde{n}, \tilde{k}$)
  $G \leftarrow$ addLeaves(G)
  $Y \leftarrow \emptyset$
  $Y' \leftarrow \emptyset$
  relevantNodes $\leftarrow$ G.leaves
  do
    $Y \leftarrow Y'$
    for q in relevantNodes **do**
      for i from 1 to $\tilde{k}+1$ **do**
        $X_i^q \leftarrow$ compute$X_i^q(G, i, q)$
      for i from 0 to $\tilde{k}$ **do**
        res $\leftarrow f_i(X_1^q, \ldots, X_{\tilde{k}+1}^q)$
        $Y' \leftarrow Y' \cup \{(v, i, q) | v \in \text{res}\}$
    interestingNodes $\leftarrow q'$ from Y for which a (v', j, q') was added to Y and q' is a non-leaf node in G with $X_l^{q'} \neq \emptyset$
    relevantNodes $\leftarrow$ getPredecessors(interestingNodes, G.predecessors)
  while $Y \neq Y'$.
  return Y

Implementation: "Less Nodes"

Dual Negative Variant

procedure FPApproximation($f, U, \tilde{n}, \tilde{k}$)
  $G \leftarrow$ emptyDualNegativeGraph($\tilde{n}, \tilde{k}$)
  $G \leftarrow$ addLeaves(G)
  $Y \leftarrow \emptyset$
  $Y' \leftarrow \emptyset$
  relevantNodes $\leftarrow$ G.leaves
  do
    $Y \leftarrow Y'$
    for q in relevantNodes **do**
      for i from 0 to $\tilde{k}$ **do**
        $\overline{X_i}^q \leftarrow$ compute$\overline{X_i}^q(G, i, q)$
      for i from 0 to $\tilde{k}$ **do**
        res $\leftarrow f_i(\overline{X_0}^q, \ldots, \overline{X_{\tilde{k}}}^q)$
        $Y' \leftarrow Y' \cup \{(v, i, q) | v \in U \setminus \text{res}\}$
    interestingNodes $\leftarrow q'$ from Y for which a (v', j, q') was added to Y and q' is a non-leaf node in G with $X_l^{q'} \neq \emptyset$
    relevantNodes $\leftarrow$ getPredecessors(interestingNodes, G.predecessors)
  while $Y \neq Y'$.
  return Y

Evaluation: Dimensions

- CPU time
- Number of iterations
- Number of nodes in the UG
- Memory usage

Evaluation: Tools

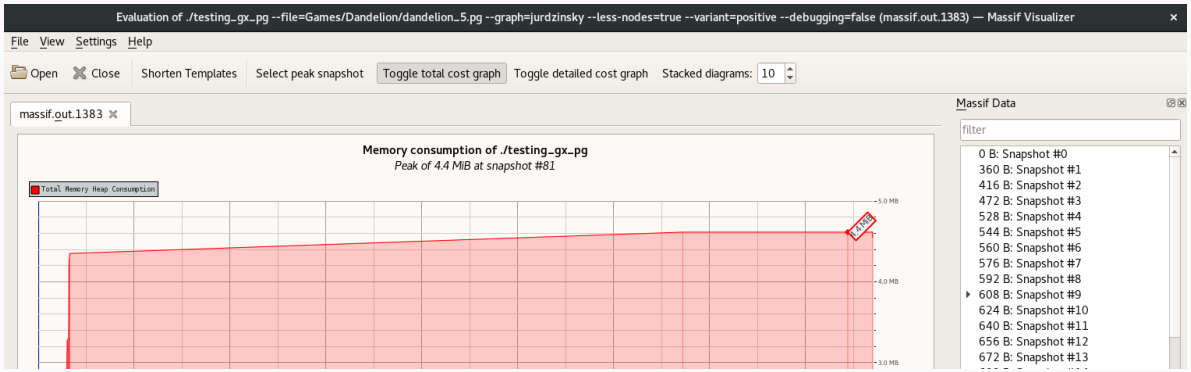
- `perf stat`

```
 Performance counter stats for './testing_gx_pg --file=Games/dandelion_5.pg --graph=jurdzinsky --less-nodes=true
--variant=positive --debugging=false' (5 runs):

             4.49 msec task-clock:u              #    0.152 CPUs utilized           ( +- 13.58% )
                0      context-switches:u        #    0.000 K/sec
                0      cpu-migrations:u          #    0.000 K/sec
              758      page-faults:u             #    0.169 M/sec                   ( +-  0.63% )
        9,071,124      cycles:u                  #    2.021 GHz                     ( +-  0.79% )
       19,041,133      instructions:u            #    2.10  insn per cycle          ( +-  0.00% )
        3,975,835      branches:u                #  885.949 M/sec                   ( +-  0.00% )
           40,046      branch-misses:u           #    1.01% of all branches         ( +-  0.16% )

          0.0296 +- 0.0250 seconds time elapsed  ( +- 84.40% )
```

- `valgrind` + `massif` + `massif-visualizer` [1]

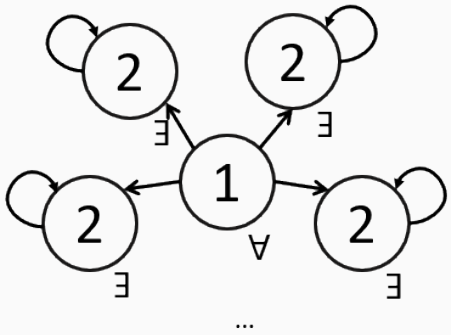


- Environment: Intel® Core™ i7-6500U CPU @ 2.50GHz, 4GB RAM

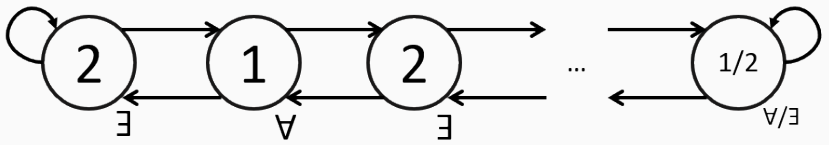
[1] `https://github.com/KDE/massif-visualizer`

Evaluation: Tasks

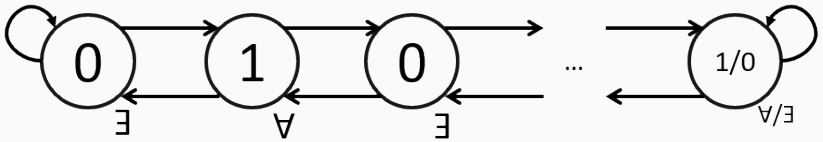
- Testing: parity games (standard, probabilistic, graded), double powerset equation system
- Performance: parity games



`Dandelion` parity games

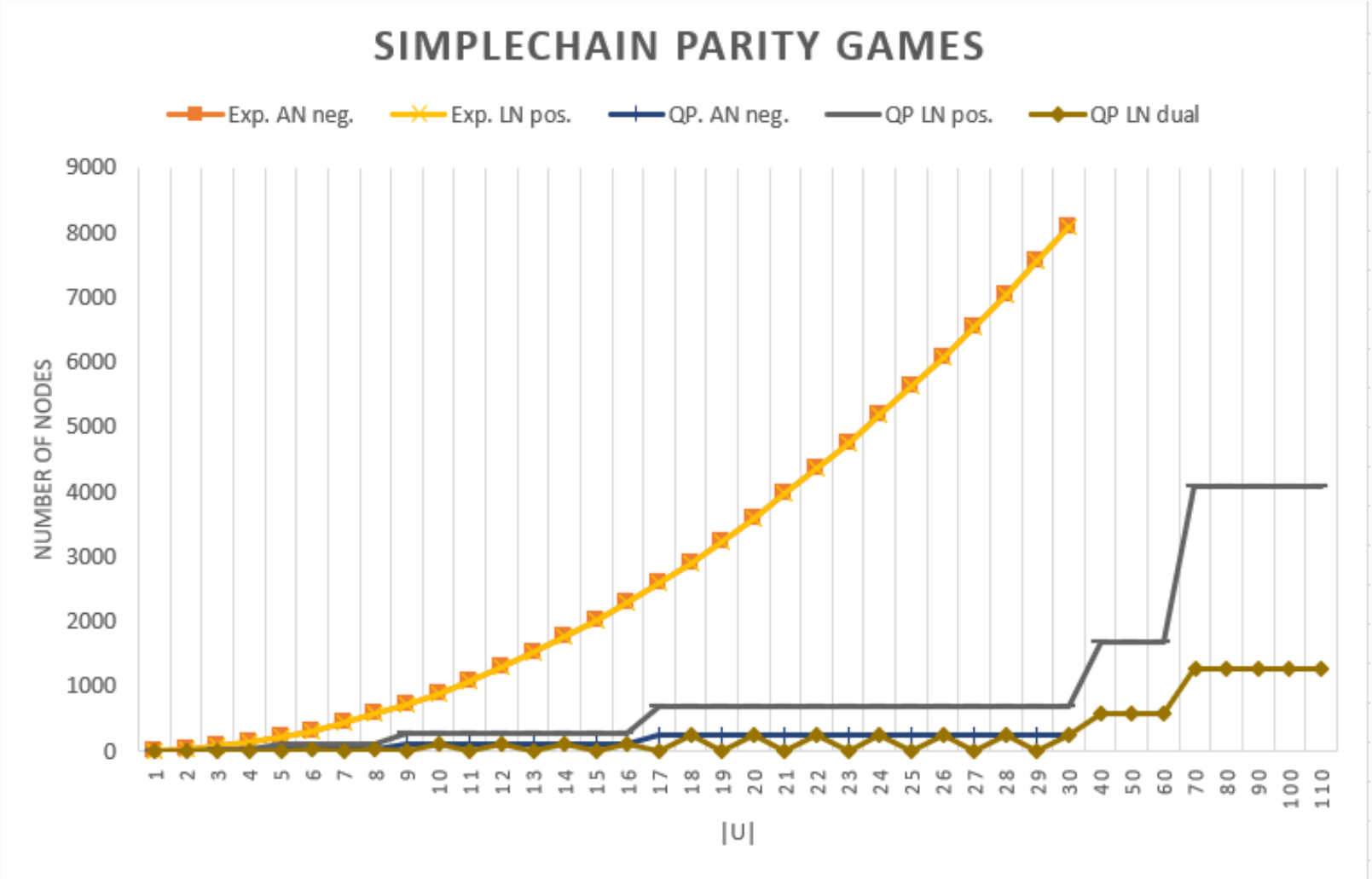


`Simplechain` parity games



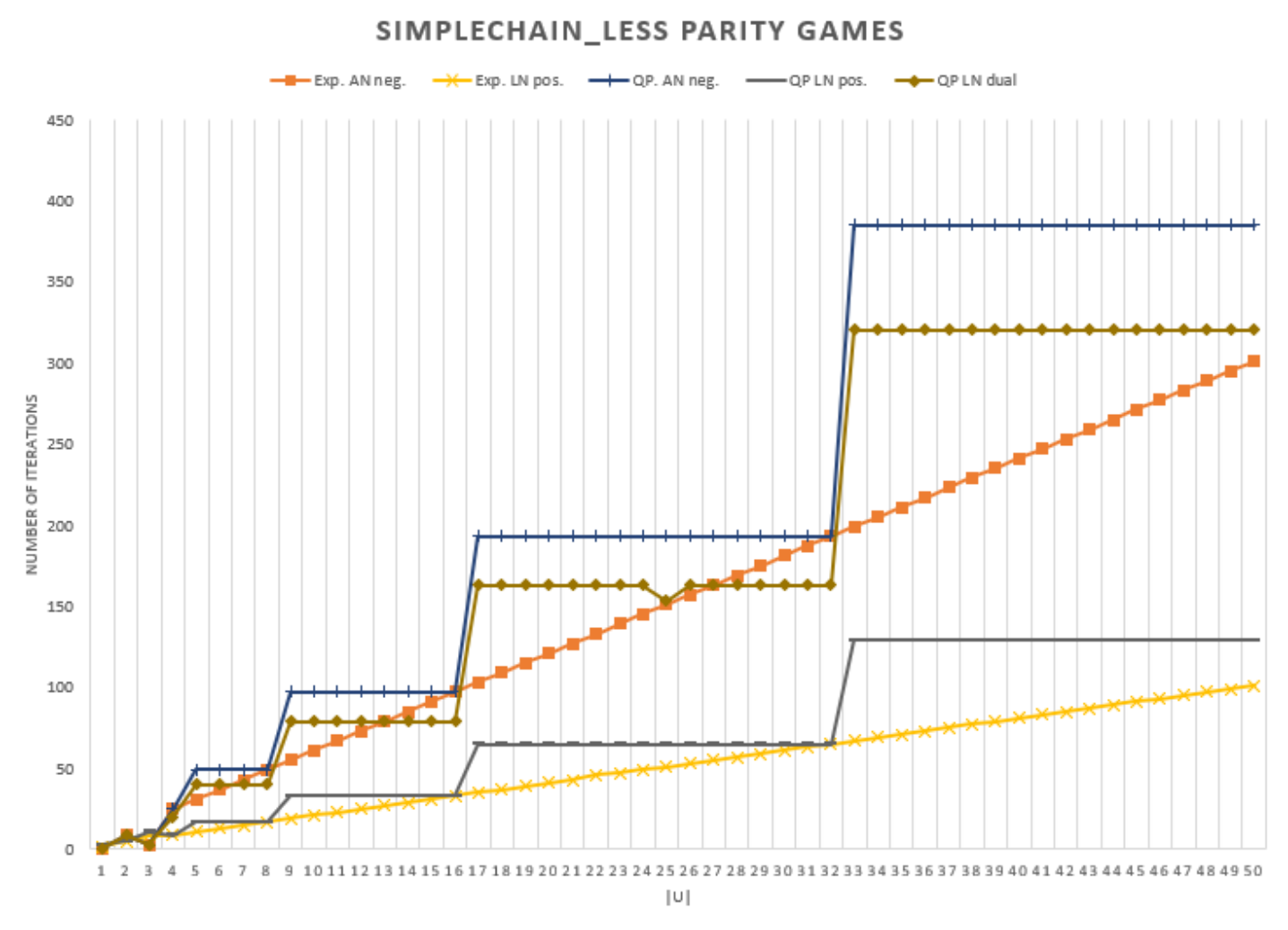
`Simplechain_less` parity games

Evaluation: Number of Nodes



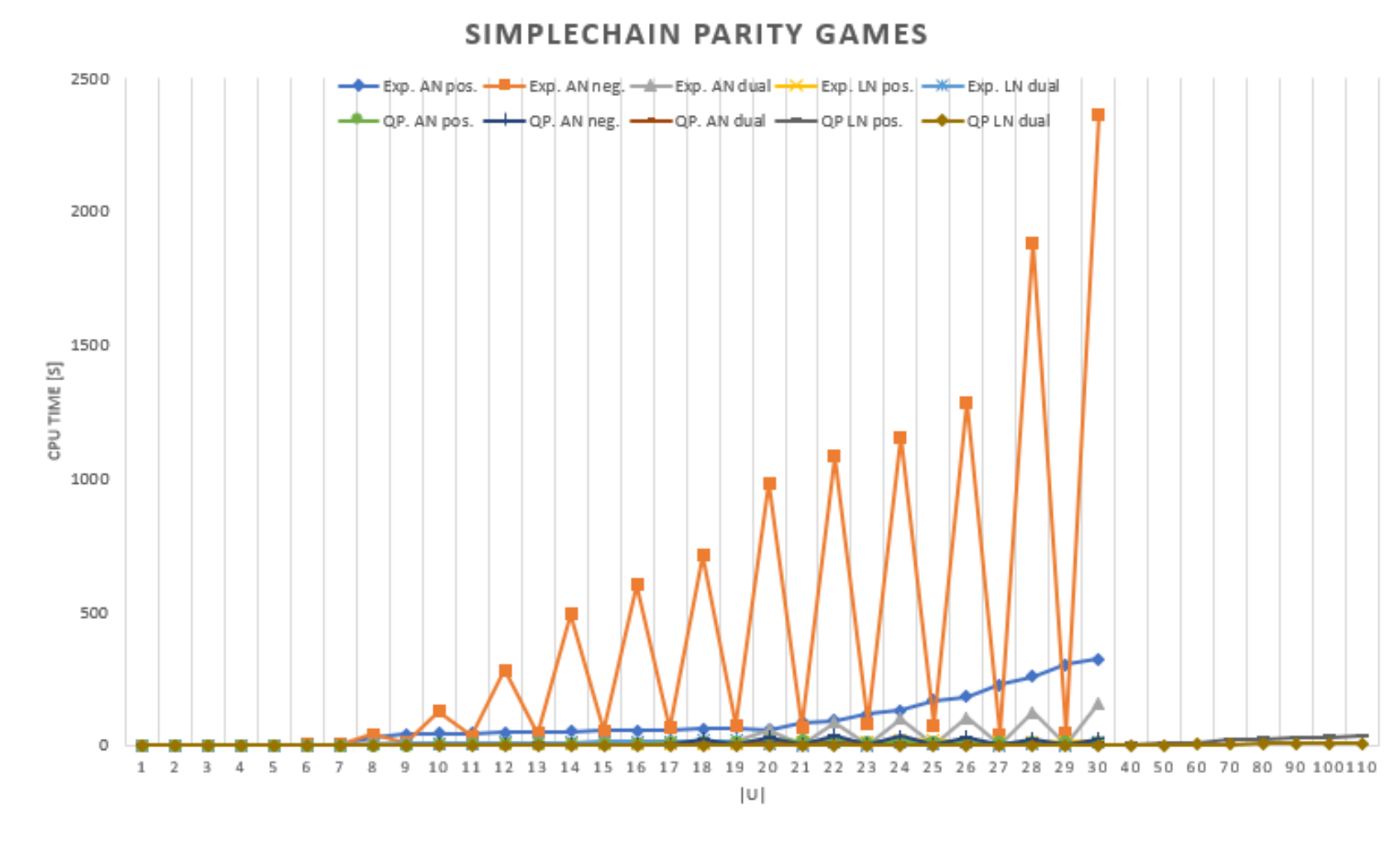
`Simplechain` parity games: number of nodes in the UG

Evaluation: Number of Iterations



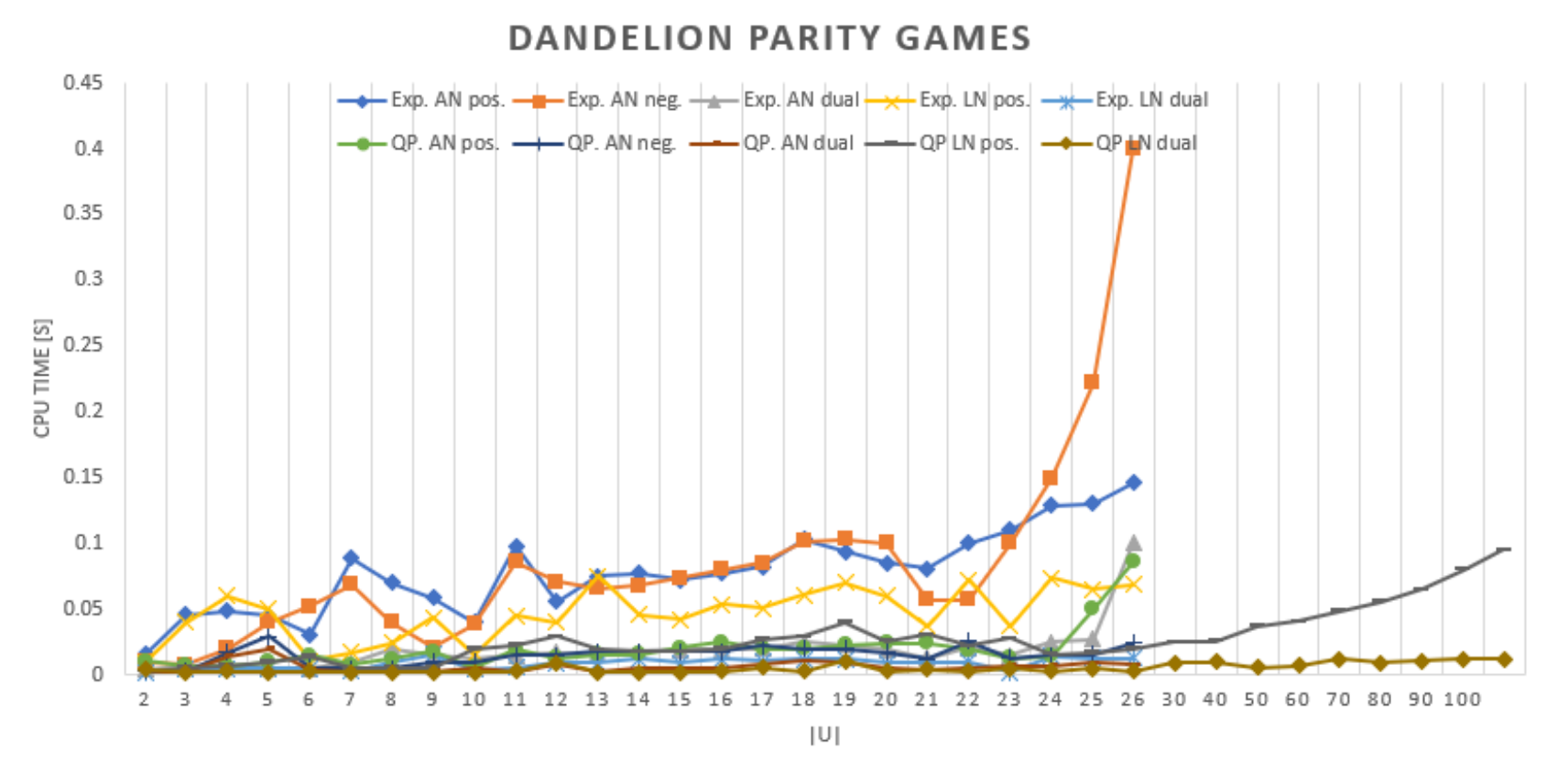
`Simplechain_less` parity games: number of iterations

Evaluation: CPU Time (1)



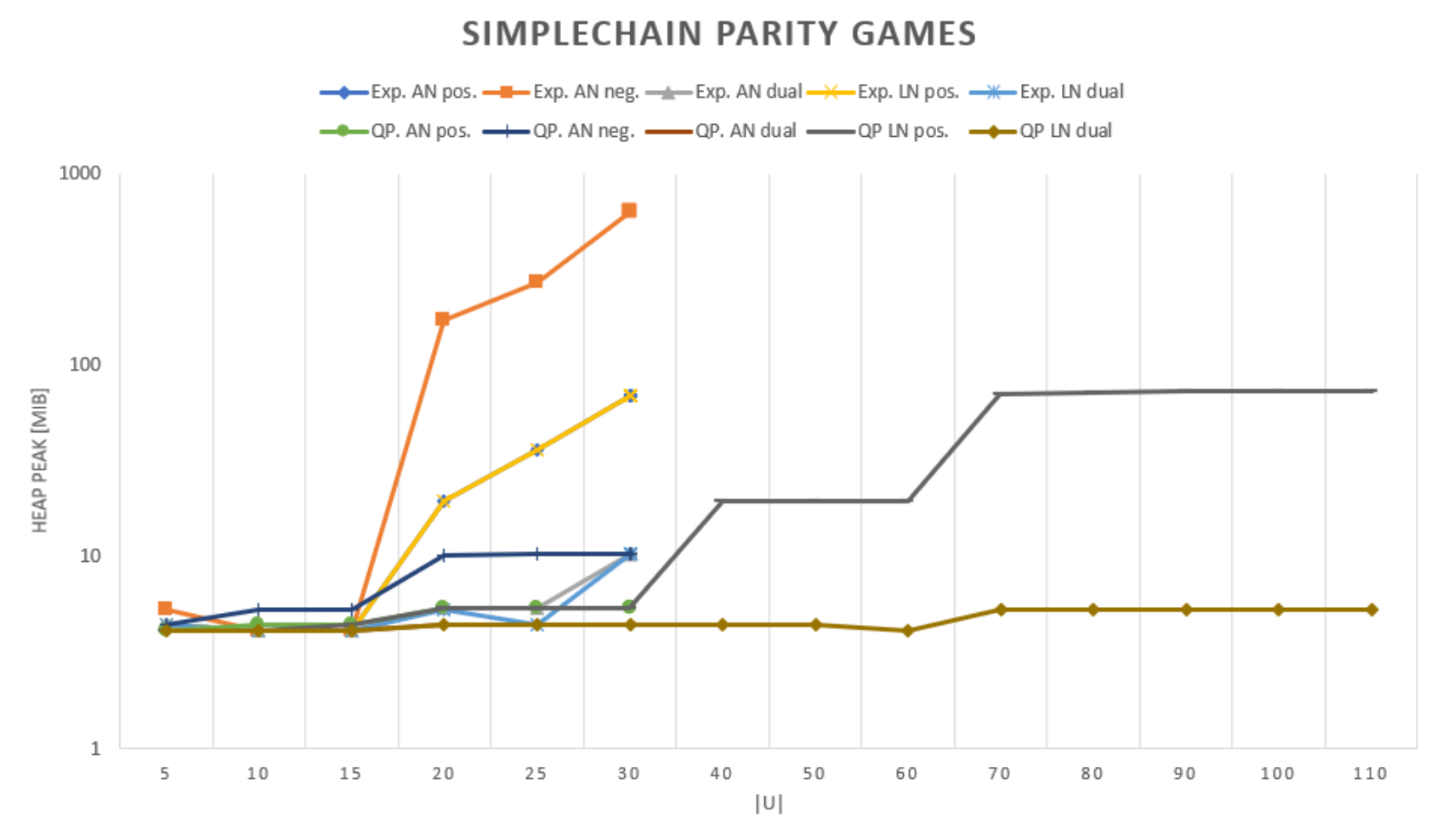
`Simplechain` parity games: CPU time

Evaluation: CPU Time (2)



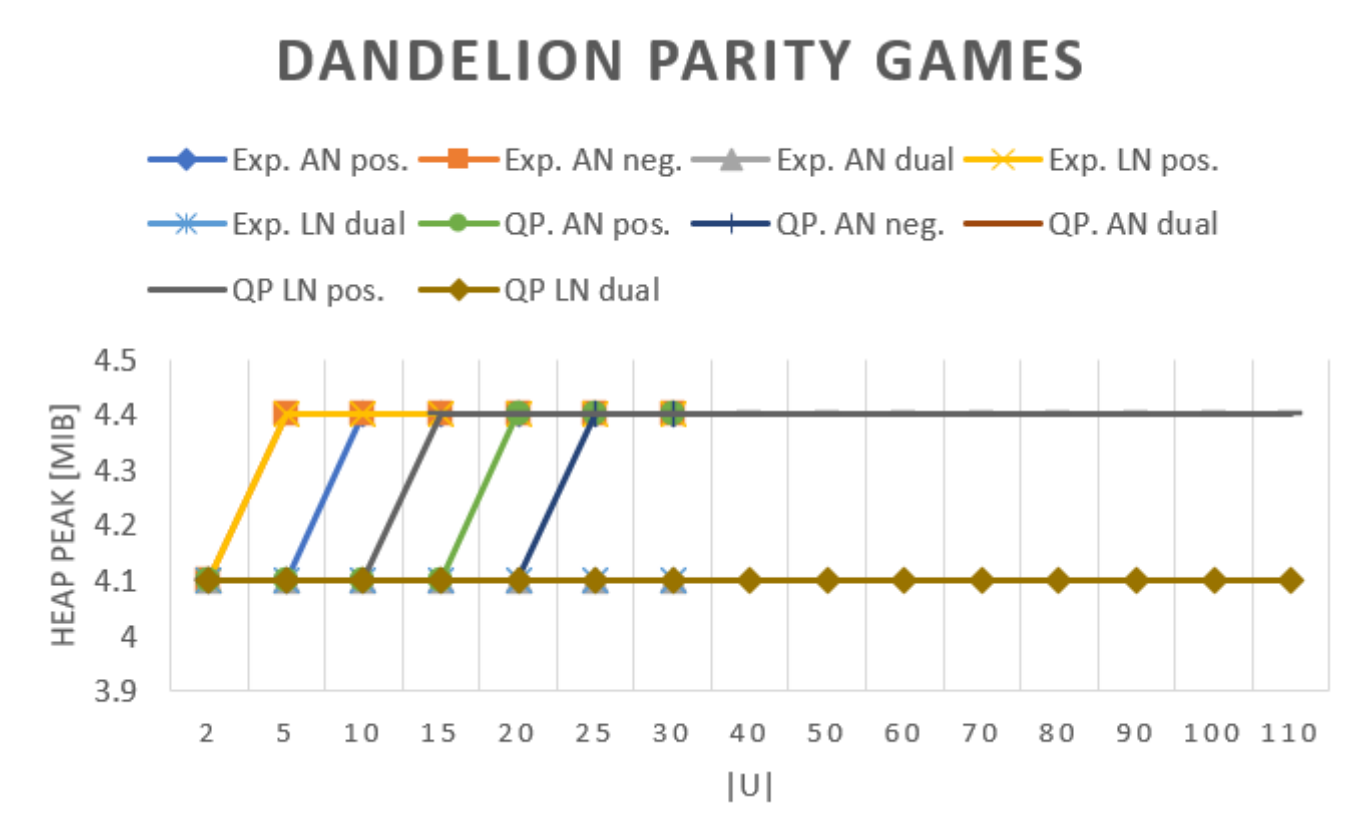
`Dandelion` parity games: CPU time

Evaluation: Memory Usage (1)



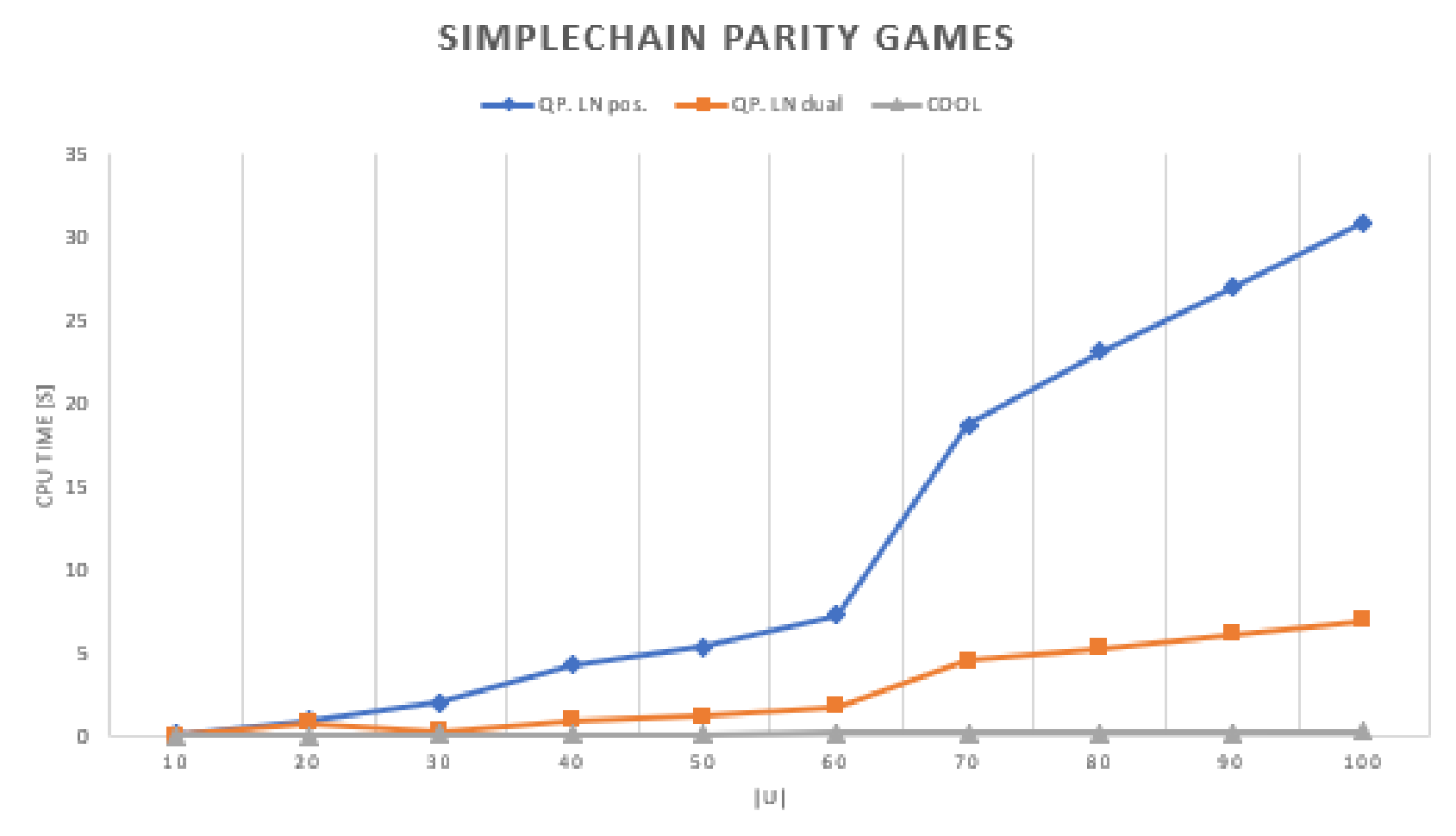
`Simplechain` parity games: Heap peak

Evaluation: Memory Usage (2)



`Dandelion` parity games: Heap peak

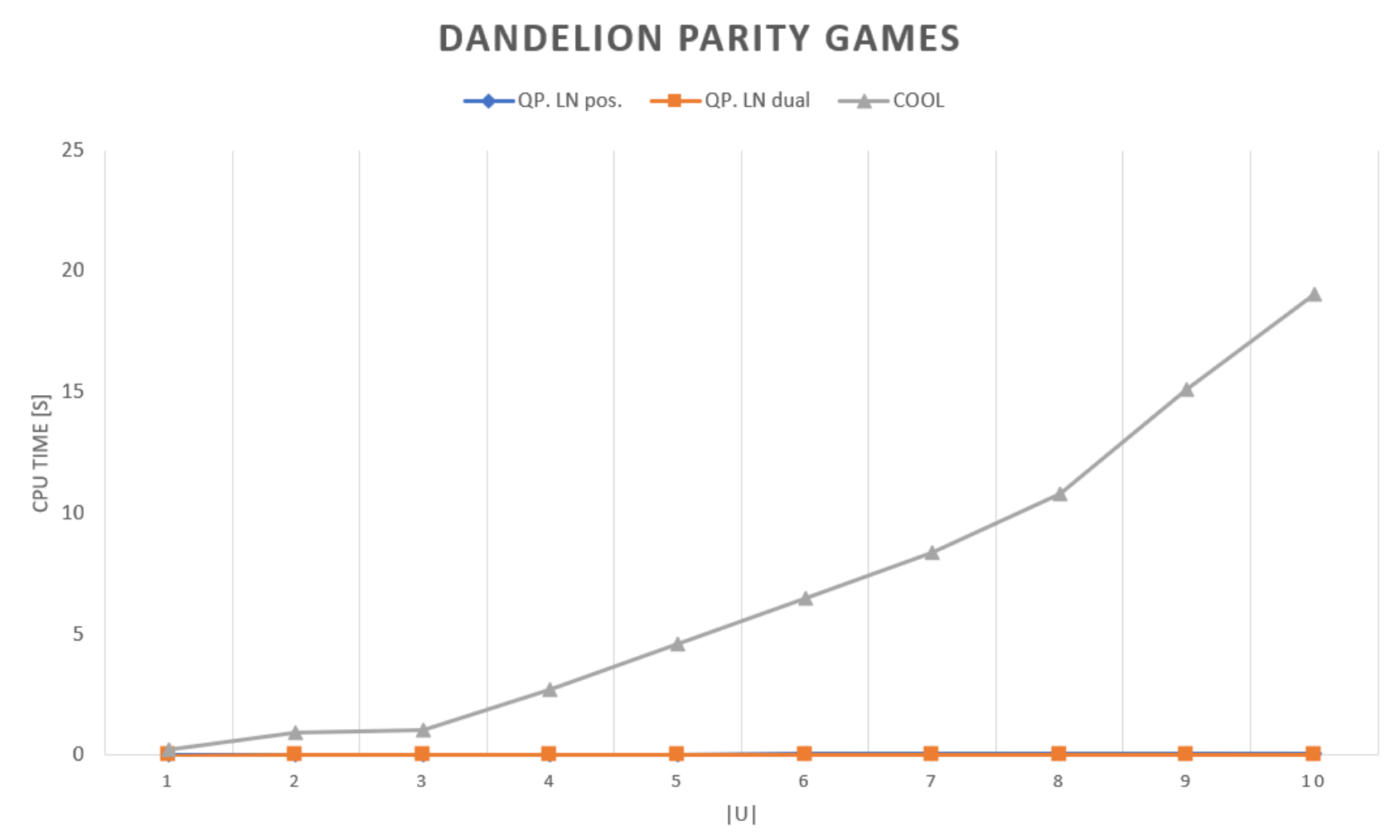
Analysis: CPU Time Comparison with COOL (1)



Simplechain parity games: Comparison of CPU time

[https://git8.cs.fau.de/software/cool/]

Analysis: CPU Time Comparison with COOL (2)



`Dandelion` parity games: Comparison of CPU time

Evaluation: Results

- **Exponential graph:** number of nodes growing fast, CPU time highly dependent on task.
- **Quasi-polynomial graph:** number of nodes growing in steps, CPU time grows steadily.
- **Number of iterations:** not directly related to number of nodes, but similar growth curve.
- **Memory usage:** dependent on n, k.

$\rightsquigarrow$ "Dual negative" variant with "less nodes" and quasi-polynomial UG has best performance over all.

Conclusion & Future Work

Conclusion:

- Implemented solver for FP equation systems.
  - Modular architecture $\rightsquigarrow$ "Designed for extension".
  - Compared different variants $\rightsquigarrow$ Use "QP LN dual negative" variant.
- Inconvenient effect on all dimensions for increasing n and k.
- Solver comparable to COOL for small n and k.
- Solver not comparable to PGSolver, but solves more general problems. [`https://github.com/tcsprojects/pgsolver`]

Conclusion & Future Work

Conclusion:

- Implemented solver for FP equation systems.
  - Modular architecture $\rightsquigarrow$ "Designed for extension".
  - Compared different variants $\rightsquigarrow$ Use "QP LN dual negative" variant.
- Inconvenient effect on all dimensions for increasing n and k.
- Solver comparable to COOL for small n and k.
- Solver not comparable to PGSolver, but solves more general problems. [`https://github.com/tcsprojects/pgsolver`]

Future work:

- Add more task modules for more equation systems.
- Integrate with COOL.
- Further improve performance.
- Implement other algorithms: Progress Measures [Jurdzinsky *et al.*, 2017], General Zielonka

Questions?

`https://git8.cs.fau.de/theses/masterarbeit-paula-welzenbach`