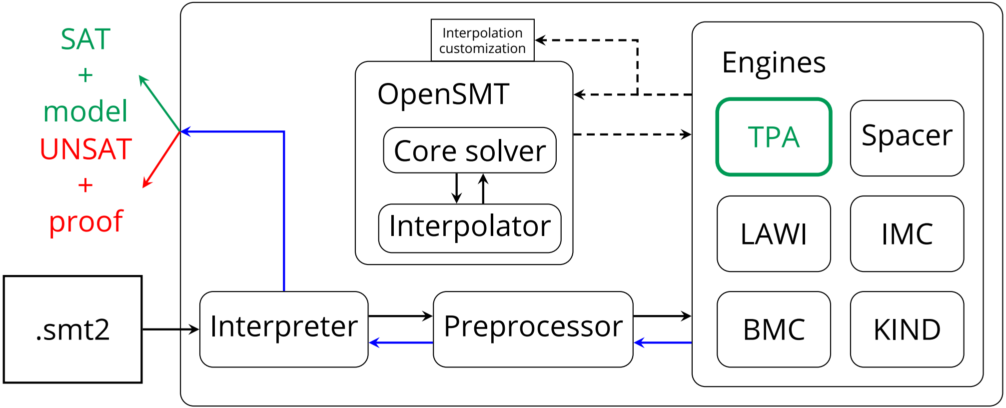
Invariants production and Transition Power Abstraction

Konstantin Britikov, Martin Blicha, Natasha Sharygina

University of Lugano, Switzerland

Golem Architecture[1]



[1] Blicha, Britikov, and Sharygina, "The Golem Horn Solver", *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part II*, 2023.

Motivation for Transition Power Abstraction (TPA)[4]

- All of the model checking engines like Spacer,[2] LAWI[3] are concentrated on states.
- Classical engines are slow in some cases (for example for deep loops).

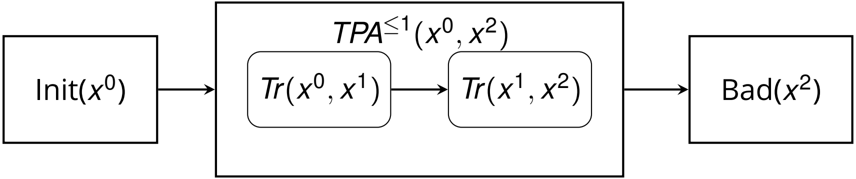
- TPA abstracts over transitions.
- TPA goes deep, finding complicated counterexamples.
- TPA turned out to be able to prove safety.

[2] Komuravelli, Gurfinkel, and Chaki, "SMT-based Model Checking For Recursive Programs", *FMSD*, 2016.
[3] McMillan, "Lazy Abstraction with Interpolants", *CAV*, 2006.
[4] Blicha et al., "Transition Power Abstractions for Deep Counterexample Detection", *TACAS*, 2022.

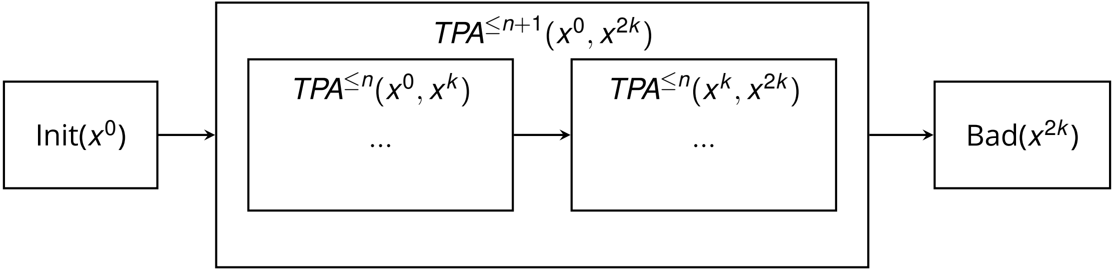
Transition Power Abstraction

$$\boxed{\mathrm{Init}(x^0)} \longrightarrow \boxed{Tr(x^0, x^1)} \longrightarrow \boxed{\mathrm{Bad}(x^1)}$$

Transition Power Abstraction



Transition Power Abstraction



Where $k = 2^n$.

Transition Power Abstraction (TPA)

Concept

Transition Power Abstraction (TPA) algorithm is based on abstract transition sequence $TPA^{\leq 0}, TPA^{\leq 1}, \ldots, TPA^{\leq n}, \ldots$

Transition Power Abstraction (TPA)

Concept

Transition Power Abstraction (TPA) algorithm is based on abstract transition sequence $TPA^{\leq 0}, TPA^{\leq 1}, \ldots, TPA^{\leq n}, \ldots$

- Overapproximates reachability up to 2^n steps of Tr
  - $Tr^i \subseteq TPA^{\leq n}$ for $0 \leq i \leq 2^n$

Transition Power Abstraction (TPA)

Concept

Transition Power Abstraction (TPA) algorithm is based on abstract transition sequence $TPA^{\leq 0}, TPA^{\leq 1}, \ldots, TPA^{\leq n}, \ldots$

- Overapproximates reachability up to 2^n steps of Tr
  - $Tr^i \subseteq TPA^{\leq n}$ for $0 \leq i \leq 2^n$
- Quantifier-free (only 2 copies of state variables)

Transition Power Abstraction (TPA)

Concept

Transition Power Abstraction (TPA) algorithm is based on abstract transition sequence $TPA^{\leq 0}, TPA^{\leq 1}, \ldots, TPA^{\leq n}, \ldots$

- Overapproximates reachability up to 2^n steps of Tr
  - $Tr^i \subseteq TPA^{\leq n}$ for $0 \leq i \leq 2^n$
- Quantifier-free (only 2 copies of state variables)
- Construction and refinement of the sequence intertwined with bounded reachability checks

Split Transition Power Abstraction (split-TPA)[5]

Concept

Adds additional checks to the TPA algorithm, now the reachability is split on two types of abstract transitions:

- $TPA^{<n+1} = TPA^{<n} \cup TPA^{=n} \circ TPA^{<n}$
- $TPA^{=n+1} = TPA^{=n} \circ TPA^{=n}$

This approach has folloving positive effects:

- Smaller, simpler checks.

[5] Blicha et al., "Split Transition Power Abstractions for Unbounded Safety", *FMCAD*, 2022.

Split Transition Power Abstraction (split-TPA)[5]

Concept

Adds additional checks to the TPA algorithm, now the reachability is split on two types of abstract transitions:

- $TPA^{<n+1} = TPA^{<n} \cup TPA^{=n} \circ TPA^{<n}$
- $TPA^{=n+1} = TPA^{=n} \circ TPA^{=n}$

This approach has folloving positive effects:

- Smaller, simpler checks.
- Both inductive and k-inductive reasoning.

[5] Blicha et al., “Split Transition Power Abstractions for Unbounded Safety”, *FMCAD*, 2022.

Split Transition Power Abstraction (split-TPA)[5]

Concept

Adds additional checks to the TPA algorithm, now the reachability is split on two types of abstract transitions:

- $TPA^{<n+1} = TPA^{<n} \cup TPA^{=n} \circ TPA^{<n}$
- $TPA^{=n+1} = TPA^{=n} \circ TPA^{=n}$

This approach has folloving positive effects:

- Smaller, simpler checks.
- Both inductive and k-inductive reasoning.
- More invariant candidates.

[5] Blicha et al., "Split Transition Power Abstractions for Unbounded Safety", *FMCAD*, 2022.

Safety checks in TPA

Both TPA and split-TPA support the production of the safety invariants. Following conditions should be satisfied for $TPA^{<n}$ to be safe inductive transition invariant:

- $Tr^i(x, x') \implies TPA^{<n}(x, x')$ for $0 \leq i < k$ and $k = 2^n$.

Safety checks in TPA

Both TPA and split-TPA support the production of the safety invariants. Following conditions should be satisfied for $TPA^{<n}$ to be safe inductive transition invariant:

- $Tr^i(x, x') \implies TPA^{<n}(x, x')$ for $0 \leq i < k$ and $k = 2^n$.
- $TPA^{<n}(x, x') \land Tr(x', x'') \implies TPA^{<n}(x, x'')$.

Safety checks in TPA

Both TPA and split-TPA support the production of the safety invariants. Following conditions should be satisfied for $TPA^{<n}$ to be safe inductive transition invariant:

- $Tr^{i}(x, x') \implies TPA^{<n}(x, x')$ for $0 \leq i < k$ and $k = 2^{n}$.
- $TPA^{<n}(x, x') \wedge Tr(x', x'') \implies TPA^{<n}(x, x'')$.
- $Init(x) \wedge TPA^{<n}(x, x') \wedge Bad(x') \implies false$.

Houdini search[6]

Houdini search - is a general algorithm to find the biggest inductive subset in a formula. Was originaly introduced to search for loop invariants.

1. We have a set of invariant candidates $C(x) = c_1(x) \wedge c_2(x) \wedge ... \wedge c_n(x)$, and some kind of transition $Tr(x, x')$.

[6] Flanagan and Leino, "Houdini, an Annotation Assistant for ESC/Java", *FME 2001: Formal Methods for Increasing Software Productivity, International Symposium of Formal Methods Europe, Berlin, Germany, March 12-16, 2001, Proceedings*, 2001.

Houdini search[6]

Houdini search - is a general algorithm to find the biggest inductive subset in a formula. Was originaly introduced to search for loop invariants.

1. We have a set of invariant candidates $C(x) = c_1(x) \wedge c_2(x) \wedge ... \wedge c_n(x)$, and some kind of transition $Tr(x, x')$.
2. $C(x) \wedge Tr(x, x') \implies C(x')$

[6] Flanagan and Leino, "Houdini, an Annotation Assistant for ESC/Java", *FME 2001: Formal Methods for Increasing Software Productivity, International Symposium of Formal Methods Europe, Berlin, Germany, March 12-16, 2001, Proceedings*, 2001.

Houdini search[6]

Houdini search - is a general algorithm to find the biggest inductive subset in a formula. Was originaly introduced to search for loop invariants.

1. We have a set of invariant candidates $C(x) = c_1(x) \land c_2(x) \land ... \land c_n(x)$, and some kind of transition $Tr(x, x')$.
2. $C(x) \land Tr(x, x') \implies C(x')$
3. true $\longrightarrow$ $C(x)$ is an invariant

[6] Flanagan and Leino, "Houdini, an Annotation Assistant for ESC/Java", *FME 2001: Formal Methods for Increasing Software Productivity, International Symposium of Formal Methods Europe, Berlin, Germany, March 12-16, 2001, Proceedings*, 2001.

Houdini search[6]

Houdini search - is a general algorithm to find the biggest inductive subset in a formula. Was originaly introduced to search for loop invariants.

1. We have a set of invariant candidates $C(x) = c_1(x) \wedge c_2(x) \wedge \ldots \wedge c_n(x)$, and some kind of transition $Tr(x, x')$.
2. $C(x) \wedge Tr(x, x') \implies C(x')$
3. true $\longrightarrow$ $C(x)$ is an invariant
4. false $\longrightarrow$ for each c_i in $C(x)$: $C(x) \wedge Tr(x, x') \implies c_i(x')$

[6] Flanagan and Leino, "Houdini, an Annotation Assistant for ESC/Java", *FME 2001: Formal Methods for Increasing Software Productivity, International Symposium of Formal Methods Europe, Berlin, Germany, March 12-16, 2001, Proceedings*, 2001.

Houdini search[6]

Houdini search - is a general algorithm to find the biggest inductive subset in a formula. Was originaly introduced to search for loop invariants.

1. We have a set of invariant candidates $C(x) = c_1(x) \wedge c_2(x) \wedge \ldots \wedge c_n(x)$, and some kind of transition $Tr(x, x')$.
2. $C(x) \wedge Tr(x, x') \implies C(x')$
3. true $\longrightarrow$ $C(x)$ is an invariant
4. false $\longrightarrow$ for each c_i in $C(x)$: $C(x) \wedge Tr(x, x') \implies c_i(x')$
5. If **4** is false $\longrightarrow$ $C := C \setminus c_i$

[6] Flanagan and Leino, "Houdini, an Annotation Assistant for ESC/Java", *FME 2001: Formal Methods for Increasing Software Productivity, International Symposium of Formal Methods Europe, Berlin, Germany, March 12-16, 2001, Proceedings*, 2001.

Houdini search[6]

Houdini search - is a general algorithm to find the biggest inductive subset in a formula. Was originaly introduced to search for loop invariants.

1. We have a set of invariant candidates $C(x) = c_1(x) \wedge c_2(x) \wedge ... \wedge c_n(x)$, and some kind of transition $Tr(x, x')$.
2. $C(x) \wedge Tr(x, x') \implies C(x')$
3. true $\longrightarrow$ $C(x)$ is an invariant
4. false $\longrightarrow$ for each c_i in $C(x)$: $C(x) \wedge Tr(x, x') \implies c_i(x')$
5. If **4** is false $\longrightarrow$ $C := C \setminus c_i$
6. After the filtering, go to step 2.

[6] Flanagan and Leino, "Houdini, an Annotation Assistant for ESC/Java", *FME 2001: Formal Methods for Increasing Software Productivity, International Symposium of Formal Methods Europe, Berlin, Germany, March 12-16, 2001, Proceedings*, 2001.

Houdini, transitions, and magic

TPA, unlike the original houdini abstracts over transitions, not states. So we had to use different approach at picking invariant candidates.

$$C(x) \wedge Tr(x, x') \implies C(x')$$

Houdini, transitions, and magic

TPA, unlike the original houdini abstracts over transitions, not states. So we had to use different approach at picking invariant candidates.

$$C(x) \wedge Tr(x, x') \implies C(x')$$

$$TPA^{<n}(x, x') \wedge Tr(x', x'') \implies TPA^{<n}(x, x'')$$

This means that we can use $TPA^{<n}(x, x')$ as a set of candidates for the intermediate invariants.

Houdini, transitions, and magic

TPA, unlike the original houdini abstracts over transitions, not states. So we had to use different approach at picking invariant candidates.

$$C(x) \wedge Tr(x, x') \implies C(x')$$

$$TPA^{<n}(x, x') \wedge Tr(x', x'') \implies TPA^{<n}(x, x'')$$

This means that we can use $TPA^{<n}(x, x')$ as a set of candidates for the intermediate invariants.

Houdini, transitions, and magic

TPA, unlike the original houdini abstracts over transitions, not states. So we had to use different approach at picking invariant candidates.

Houdini, transitions, and magic

TPA, unlike the original houdini abstracts over transitions, not states. So we had to use different approach at picking invariant candidates.

$$TrInv(x, x') \subseteq TPA^{<n}(x, x')$$

Usage of invariants

Invariants produced by the usage of Houdini algorithm can be utilised in solving to refine the abstraction. As you recall, this are three main points in proving safety:

Usage of invariants

Invariants produced by the usage of Houdini algorithm can be utilised in solving to refine the abstraction. As you recall, this are three main points in proving safety:

1. $Tr^i(x, x') \implies TPA^{<n}(x, x')$ for $0 \leq i < k$ and $k = 2^n$.

Usage of invariants

Invariants produced by the usage of Houdini algorithm can be utilised in solving to refine the abstraction. As you recall, this are three main points in proving safety:

1. $Tr^i(x, x') \implies TPA^{<n}(x, x')$ for $0 \leq i < k$ and $k = 2^n$.
2. $TPA^{<n}(x, x') \wedge Tr(x', x'') \implies TPA^{<n}(x, x'')$

Usage of invariants

Invariants produced by the usage of Houdini algorithm can be utilised in solving to refine the abstraction. As you recall, this are three main points in proving safety:

1. $Tr^i(x, x') \implies TPA^{<n}(x, x')$ for $0 \leq i < k$ and $k = 2^n$.
2. $TPA^{<n}(x, x') \wedge Tr(x', x'') \implies TPA^{<n}(x, x'')$
3. $Init(x) \wedge TPA^{<n}(x, x') \wedge Bad(x') \implies false$

Usage of invariants

Invariants produced by the usage of Houdini algorithm can be utilised in solving to refine the abstraction. As you recall, this are three main points in proving safety:

1. $Tr^i(x, x') \implies TPA^{<n}(x, x')$ for $0 \leq i < k$ and $k = 2^n$.
2. $TPA^{<n}(x, x') \wedge Tr(x', x'') \implies TPA^{<n}(x, x'')$
3. $Init(x) \wedge TPA^{<n}(x, x') \wedge Bad(x') \implies false$

1. $Tr^i(x, x') \implies TPA^{<n}(x, x')$ for $0 \leq i < k$ and $k = 2^n$.

Usage of invariants

Invariants produced by the usage of Houdini algorithm can be utilised in solving to refine the abstraction. As you recall, this are three main points in proving safety:

1. $Tr^i(x, x') \implies TPA^{<n}(x, x')$ for $0 \leq i < k$ and $k = 2^n$.
2. $TPA^{<n}(x, x') \wedge Tr(x', x'') \implies TPA^{<n}(x, x'')$
3. $Init(x) \wedge TPA^{<n}(x, x') \wedge Bad(x') \implies false$

1. $Tr^i(x, x') \implies TPA^{<n}(x, x')$ for $0 \leq i < k$ and $k = 2^n$.
2. $TPA^{<n}(x, x') \wedge Tr(x', x'') \wedge TrInv(x, x'') \implies TPA^{<n}(x, x'')$

Usage of invariants

Invariants produced by the usage of Houdini algorithm can be utilised in solving to refine the abstraction. As you recall, this are three main points in proving safety:

1. $Tr^i(x, x') \implies TPA^{<n}(x, x')$ for $0 \leq i < k$ and $k = 2^n$.
2. $TPA^{<n}(x, x') \wedge Tr(x', x'') \implies TPA^{<n}(x, x'')$
3. $Init(x) \wedge TPA^{<n}(x, x') \wedge Bad(x') \implies false$

1. $Tr^i(x, x') \implies TPA^{<n}(x, x')$ for $0 \leq i < k$ and $k = 2^n$.
2. $TPA^{<n}(x, x') \wedge Tr(x', x'') \wedge TrInv(x, x'') \implies TPA^{<n}(x, x'')$
3. $Init(x) \wedge TPA^{<n}(x, x') \wedge TrInv(x, x') \wedge Bad(x') \implies false$

Evaluation

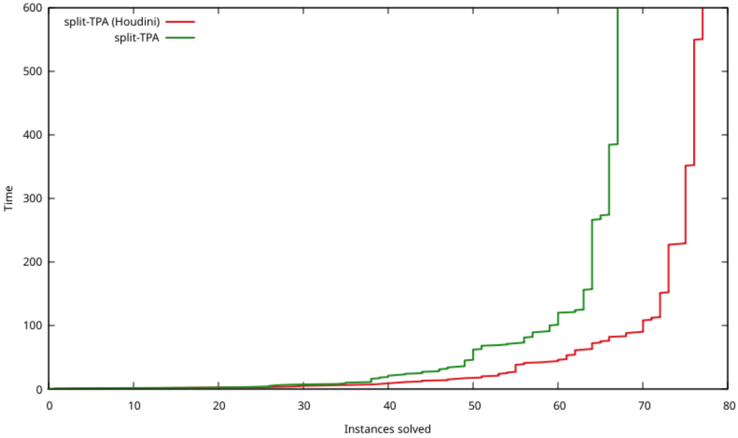
Comparison of Houdini and Non-Houdini performance (600 seconds timeout)

	Number of tests	split-TPA (Houdini)	split-TPA	TPA (Houdini)	TPA
LIA linear	585	342	332	302	295
LRA linear	498	202	196	136	130

Table: CHC-COMP'21 selection

Evaluation

Comparison of Houdini and Non-Houdini performance (600 seconds timeout)



Future Work

- Introduce Houdini-based invariant search in exact transition abstractions
- Filter out low-potential candidates
- Improvements to the algorithm to pick the candidates for invariants

Conclusion

Houdini application to the TPA

- Was able to improve performance of the split-TPA and TPA

Conclusion

Houdini application to the TPA

- Was able to improve performance of the split-TPA and TPA
- Is open-source `github.com/usi-verification-and-security/golem`

Conclusion

Houdini application to the TPA

- Was able to improve performance of the split-TPA and TPA
- Is open-source `github.com/usi-verification-and-security/golem`
- Searching for PhDs and PostDocs

Conclusion

Houdini application to the TPA

- Was able to improve performance of the split-TPA and TPA
- Is open-source `github.com/usi-verification-and-security/golem`
- Searching for PhDs and PostDocs
- Check out our website: https://verify.inf.usi.ch/

Questions?

References I

Blicha, Martin, Konstantin Britikov, and Natasha Sharygina. "The Golem Horn Solver". In: *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part II*. Ed. by Constantin Enea and Akash Lal. Vol. 13965. Lecture Notes in Computer Science. Springer, 2023, pp. 209–223. doi: `10.1007/978-3-031-37703-7\_10`. url: `https://doi.org/10.1007/978-3-031-37703-7\_10`.

Blicha, Martin et al. "Split Transition Power Abstractions for Unbounded Safety". In: *FMCAD*. Ed. by Alberto Griggio and Neha Rungta. Cham: TU Wien Academic Press, 2022, pp. 349–358. doi: `10.34727/2022/isbn.978-3-85448-053-2_42`.

— ."Transition Power Abstractions for Deep Counterexample Detection". In: *TACAS*. Ed. by Dana Fisman and Grigore Rosu. Cham: Springer International Publishing, 2022, pp. 524–542.

References II

- Flanagan, Cormac and K. Rustan M. Leino. "Houdini, an Annotation Assistant for ESC/Java". In: *FME 2001: Formal Methods for Increasing Software Productivity, International Symposium of Formal Methods Europe, Berlin, Germany, March 12-16, 2001, Proceedings*. Ed. by José Nuno Oliveira and Pamela Zave. Vol. 2021. Lecture Notes in Computer Science. Springer, 2001, pp. 500–517. doi: `10.1007/3-540-45251-6\_29`. url: `https://doi.org/10.1007/3-540-45251-6\_29`.
- Komuravelli, Anvesh, Arie Gurfinkel, and Sagar Chaki. "SMT-based Model Checking For Recursive Programs". In: *FMSD*. Vol. 48. 3. 2016, pp. 175–205.
- McMillan, Kenneth L. "Lazy Abstraction with Interpolants". In: *CAV*. Ed. by Thomas Ball and Robert B. Jones. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 123–136.