Certifying Deadlock Freedom of BIP Models

A Case Study for the Certification of Safety Critical Software

Jan Olaf Blech

Overview

- **Our General Methodology**
- **The D-Finder Case Study**
- **Improvements and Future Work**

Our Approach

- **Goal: guarantee correctness of verification tools**

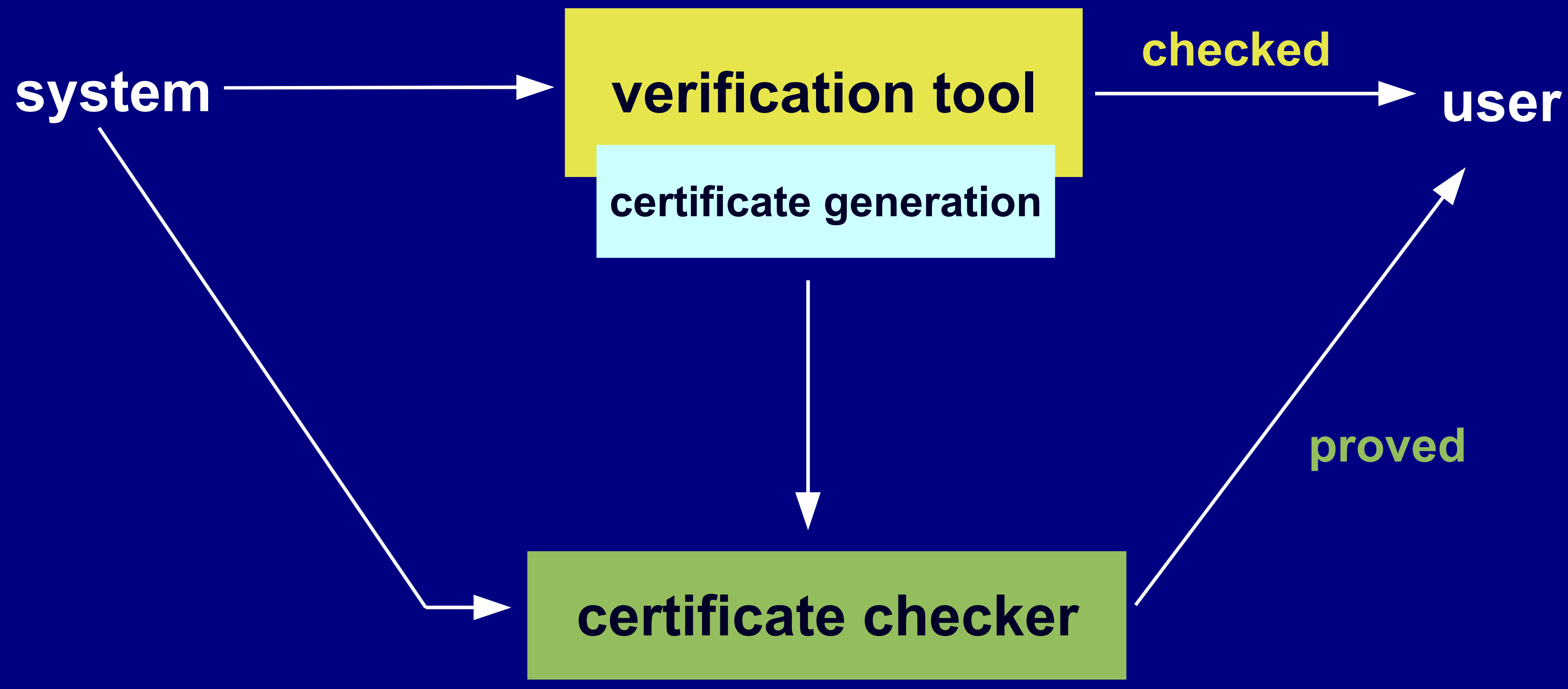
Our Approach

- **Goal: guarantee correctness of verification tools**

- verification tools may contain errors
  - wrong results
  - not accepted by certification authorities
- verification tools are often domain specific
  - results can be difficult to understand
  - reuse by other verification tools can be hard

Our Approach

- **verification tools generate certificates**

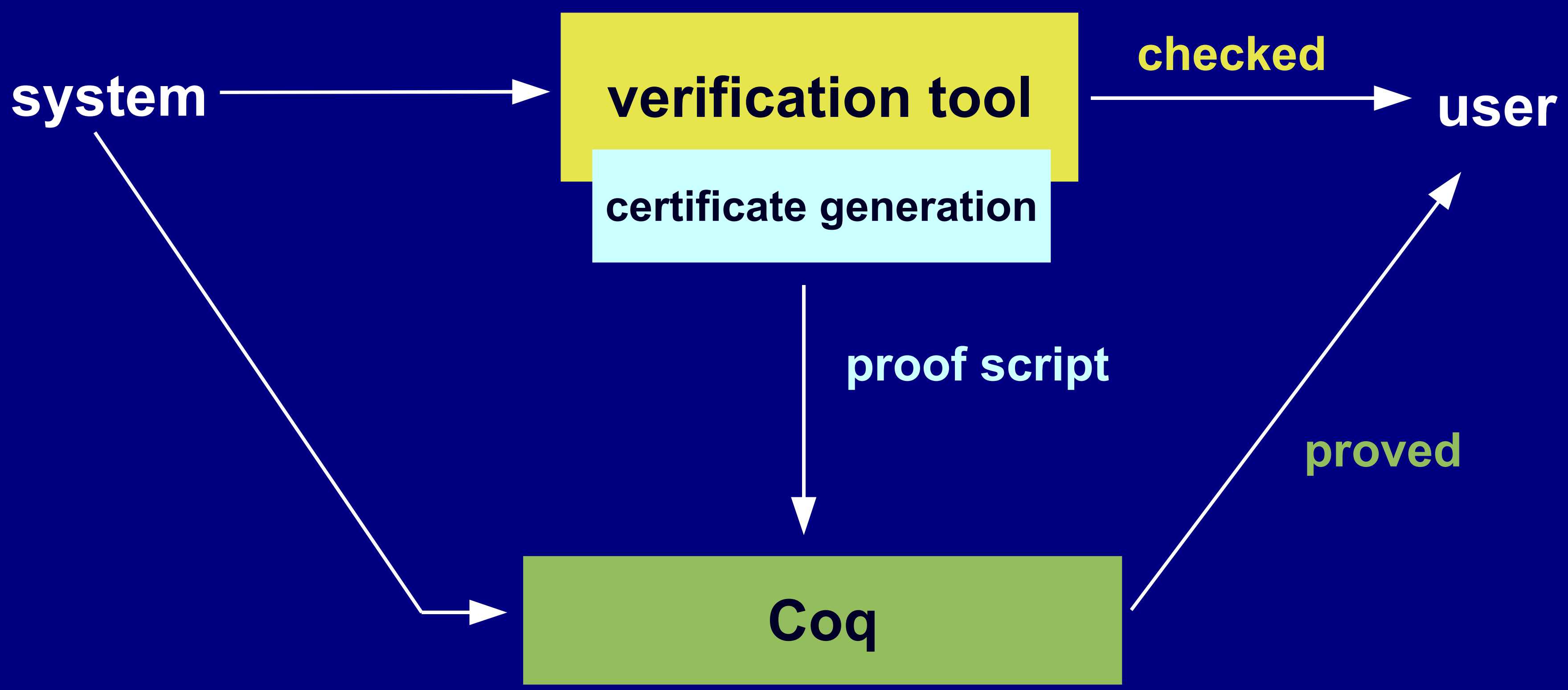


Main Idea

- Automated verification tools (e.g. model checkers)
  - relatively fast / high degree of automation
  - specific application domain
  - large untrusted code base
- Higher-order theorem provers (e.g. Coq)
  - relatively slow / interactive reasoning
  - can be used for all kinds of logical reasoning
  - high level of trust
- Combine the advantages

Our Approach

- **verification tools generate certificates**



Main Characteristics

- results of automated verification tools are put to a high level of trust
  - for certification of software systems
    - Common Criteria EAL 7 certification
  - without having to reveal verification tool know-how
    - robust to undocumented extensions
  - by using human readable specifications
    - formalized in a higher-order theorem prover

Main Characteristics

- certificates are theorem prover proof scripts
  - certificate: property + proof
  - creation by just documenting the discovery process
    - no need to redo tasks that have been done by the verification tool
    - robust to minor implementation changes
    - relatively easy -- "intelligent part" is in the algorithms of the tool
  - general interchange format
  - allows for combination of certificates
  - checking them may be a bottleneck

Main Characteristics

- certificates are theorem prover proof scripts
  - certificate: property + proof

    main challenge for the verification tool developer

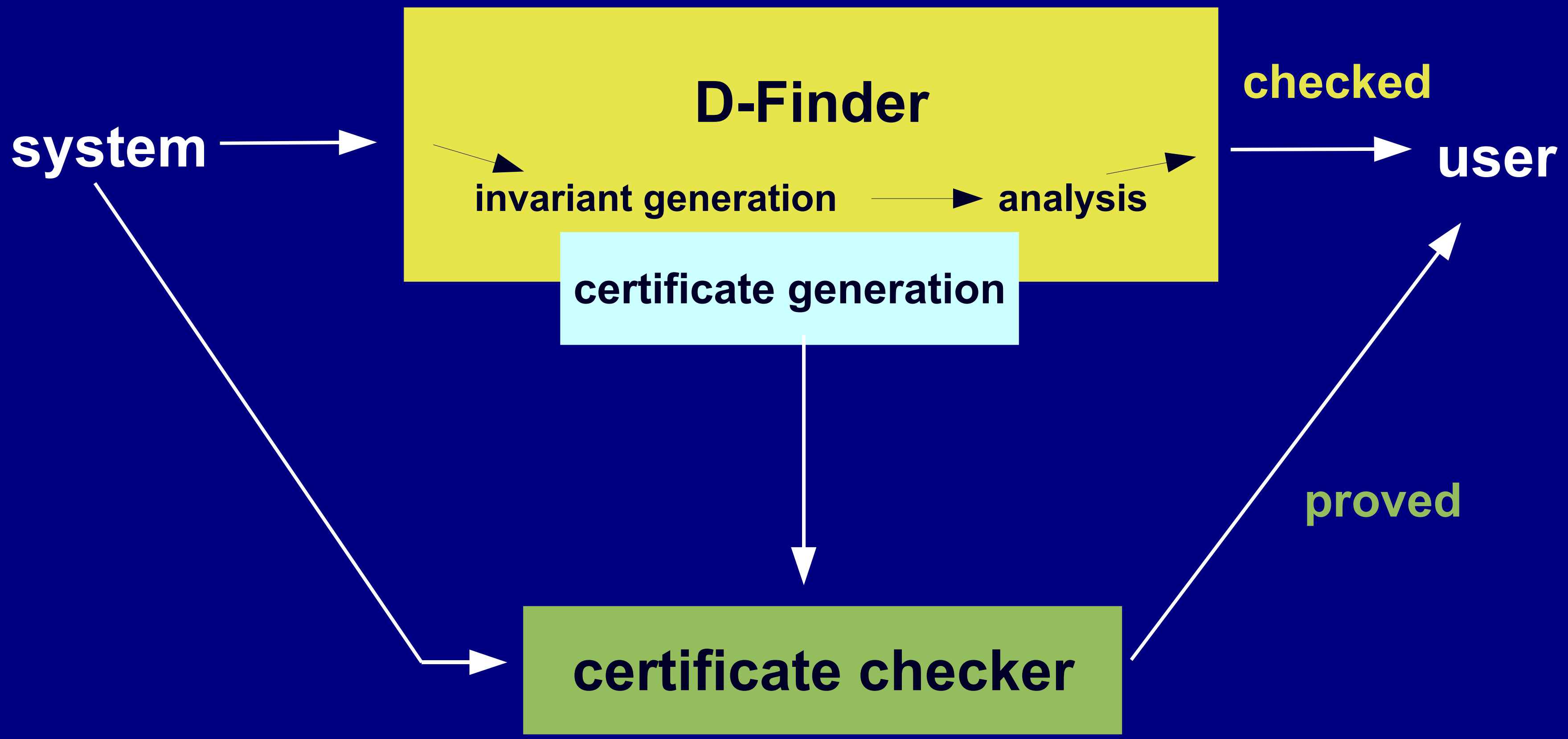
  - creation by just documenting the discovery process
    - no need to redo tasks that have been done by the verification tool
    - robust to minor implementation changes
    - relatively easy -- "intelligent part" is in the algorithms of the tool
  - general interchange format
  - allows for combination of certificates
  - checking them may be a bottleneck

main challenge for the certificate infrastructure developer

Overview

- Our General Methodology
- The D-Finder Case Study
- Improvements and Future Work

Certificates for D-Finder



Certificates for D-Finder

the generated proofs

$$\forall s.\ \mathrm{Reachable}_{BM}(s) \rightarrow \mathrm{Enabled}_{BM}(s)$$

↑

$$\forall s.\ \mathrm{Reachable}_{BM}(s) \rightarrow \neg \mathrm{DIS}_{BM}(s)$$

↑

$$\forall s.\ \mathrm{Reachable}_{BM}(s) \rightarrow II(s) \wedge CI(s) \qquad \forall s.\ II(s) \wedge CI(s) \rightarrow \neg \mathrm{DIS}_{BM}(s)$$

↑

$$\forall s.\ \mathrm{Reachable}_{BM}(s) \rightarrow II(s) \wedge CI(s) \qquad \forall s.\ \neg(II(s) \wedge CI(s) \wedge \mathrm{DIS}_{BM}(s))$$

Certificates for D-Finder

no deadlocks

$$\forall\, s.\ Reachable_{BM}(s) \rightarrow Enabled_{BM}(s)$$

↑

$$\forall\, s.\ Reachable_{BM}(s) \rightarrow \neg DIS_{BM}(s)$$

↑

$$\forall\, s.\ Reachable_{BM}(s) \rightarrow II(s) \wedge CI(s) \qquad \forall\, s.\ II(s) \wedge CI(s) \rightarrow \neg DIS_{BM}(s)$$

↑

$$\forall\, s.\ Reachable_{BM}(s) \rightarrow II(s) \wedge CI(s) \qquad \forall\, s.\ \neg(II(s) \wedge CI(s) \wedge DIS_{BM}(s))$$

Certificates for D-Finder

$\forall$ s. Reachable_{BM} (s) $\rightarrow$ Enabled_{BM} (s)

↑

$\forall$ s. Reachable_{BM}(s) $\rightarrow$ $\neg\text{DIS}_{BM}$(s)

↑

$\forall$ s. Reachable_{BM}(s) $\rightarrow$ II(s) $\wedge$ CI(s)

$\forall$ s. II(s) $\wedge$ CI(s) $\rightarrow$ $\neg\text{DIS}_{BM}$(s)

↑

$\forall$ s. Reachable_{BM}(s) $\rightarrow$ II(s) $\wedge$ CI(s)

$\forall$ s. $\neg$(II(s) $\wedge$ CI(s) $\wedge$ DIS_{BM}(s))

most challenging task

Coq Semantics

Operational semantics for flat BIP models

- atomic components
  - states
    - variables: (var $\Rightarrow$ val) mapping
    - location
  - transitions
    - source location
    - guard function: (var $\Rightarrow$ val) $\Rightarrow$ bool
    - update function: (var $\Rightarrow$ val) $\Rightarrow$ (var $\Rightarrow$ val)
    - port
    - target location
- composed components
  - states: list of atomic components' states
  - interactions: list of ports
  - semantics: 1 inference rule

Proving an Inductive Invariant

- verification goal

$$\forall s.\ Reachable_{BM}(s) \rightarrow$$

$$CI_1(s) \wedge \ldots \wedge CI_n(s) \wedge II_1(s) \wedge \ldots \wedge II_m(s)$$

$$a_1(s) \vee \ldots \vee a_j(s)$$

prove subpredicates independently

Proving an Inductive Invariant

- induction

$\Rightarrow$ $(a_1(init) \vee \ldots \vee a_n(init))$

$\Rightarrow$ $(a_1(s) \vee \ldots \vee a_n(s))$

$$\frac{s \rightarrow s'}{(a_1(s') \vee \ldots \vee a_n(s'))}$$

Proving an Inductive Invariant

- induction

⇨ $(a_1(init) \vee \ldots \vee a_n(init))$

predicates may not always be inductive

⇨ $(a_1(s) \vee \ldots \vee a_n(s))$

$$\frac{s \rightarrow s'}{(a_1(s') \vee \ldots \vee a_n(s'))}$$

Proving an Inductive Invariant

Solution 1: strengthening using a predicate C

- induction

⇨ $(a_1(init) \vee \ldots \vee a_n(init)) \wedge C(init)$

⇨ $(a_1(s) \vee \ldots \vee a_n(s)) \wedge C(s)$

$$\frac{s \rightarrow s'}{(a_1(s') \vee \ldots \vee a_n(s')) \wedge C(s')}$$

Proving an Inductive Invariant

Solution 2: produce inductive invariants via robust BIP models

- idea: make invariants weaker so that they become inductive
  - some parts of invariants seemed artificial/unnatural
  - some values are delivered by sensors
    - they have a certain range of unpreciseness
    - keep this range in the BIP model and generate invariants for these models $\Rightarrow$ tend to be inductive
    - generated invariants are invariants of the original BIP model

[Blech + Nguyen + Périn '09]

Evaluation

- what has been implemented
  - (subset of) BIP semantics for Coq
  - Coq representation generation implemented in Java for (a subset of) BIP based on Java library for BIP2
  - automatic proof script generation for invariants based on invariants provided by D-Finder
    - implemented in Ocaml
    - needs some manual instantiation for certain guard and update expressions
- a few minutes checking time for small BIP models

Overview

- **Our General Methodology**
- **The D-Finder Case Study**
- **Improvements and Future Work**

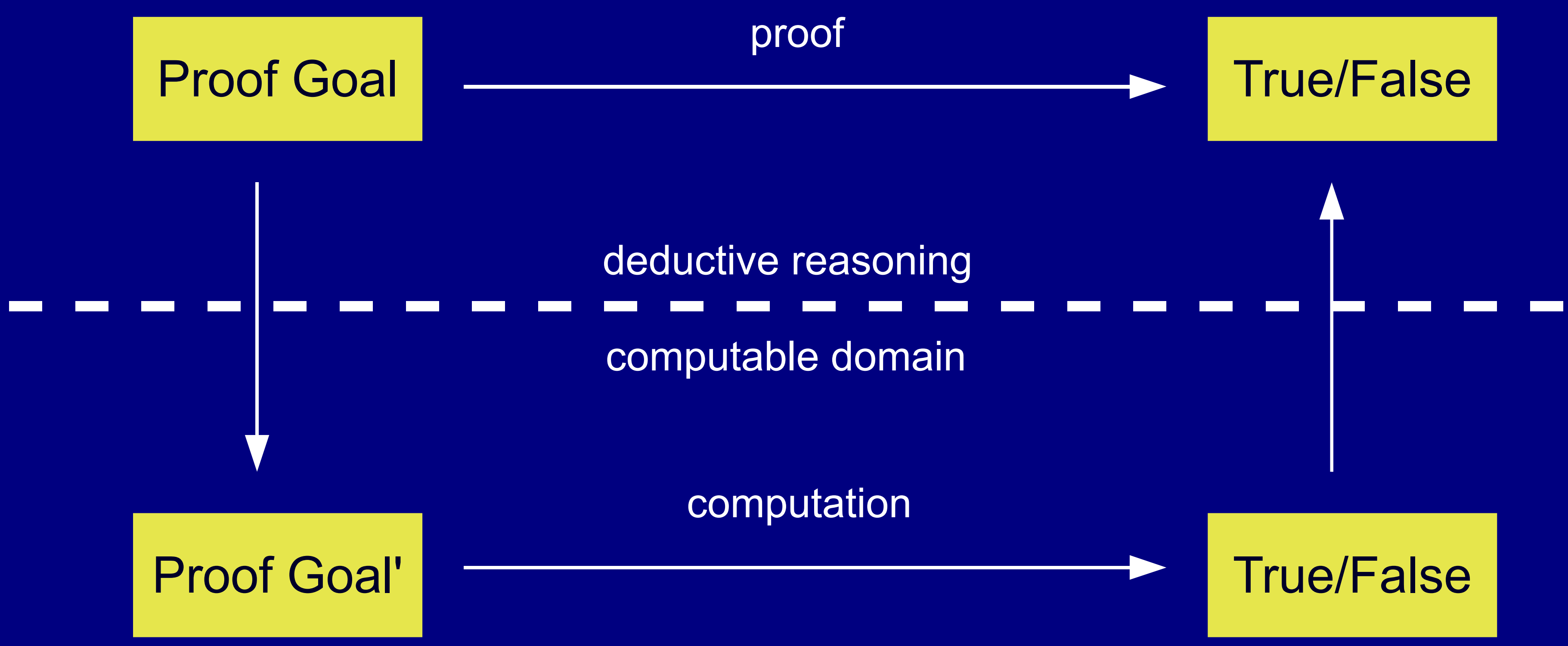
Reducing Certificate Checking time by using Checker Predicates

- Idea:
  - higher-order theorem provers are slow

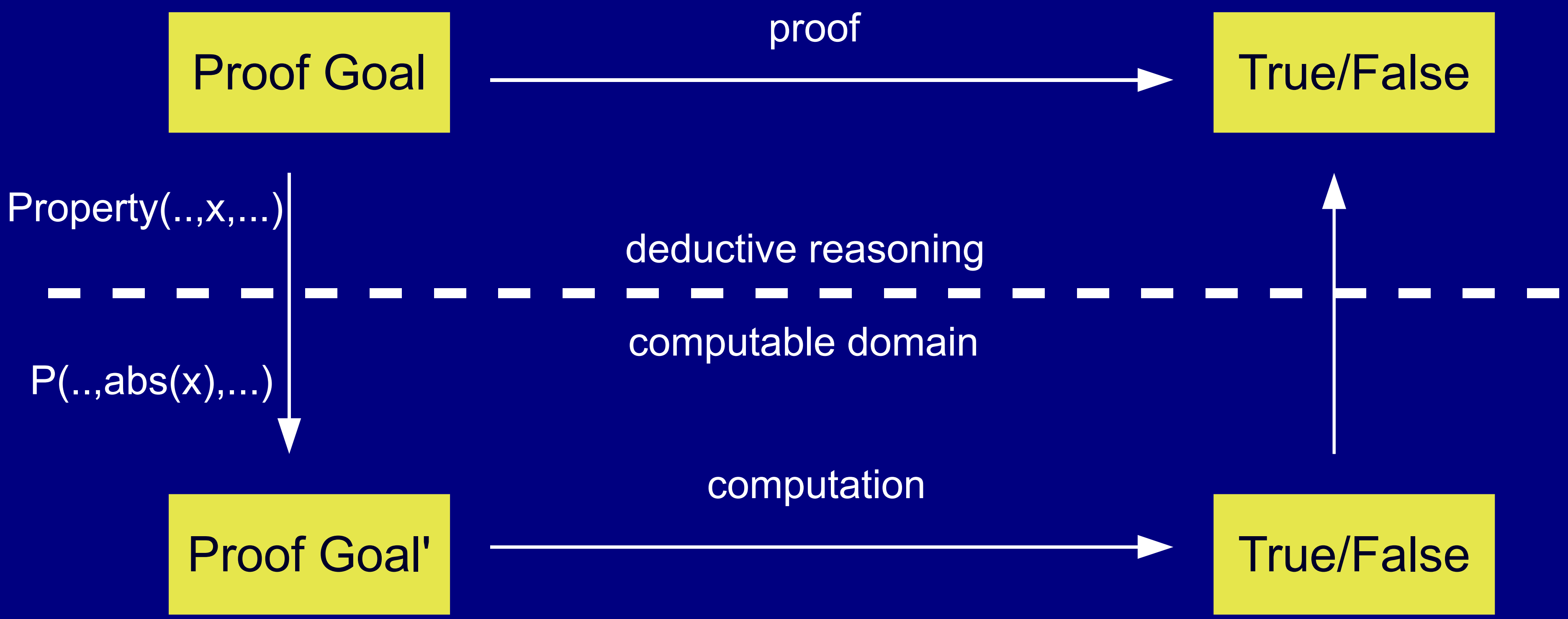
    most proof scripts require
    - some search for proofs using tactics
    - deductive reasoning
    - higher-order unfications
  - replace this by something computable

[joint work with Benjamin Grégoire, INRIA]

Reducing Certificate Checking time by using Checker Predicates



Reducing Certificate Checking time by using Checker Predicates



Checker Predicates

are predicates formalized in a theorem prover

- take e.g. program representations, state representations as input
- equivalence or implication of non-checker specification
  - used instead of tactic applications
  - direct use out of a proof script
  - require correctness proof
- are formalized in an executable way
  - no expensive unifications and rewritings
  - speeds proving process up

Checker Predicates

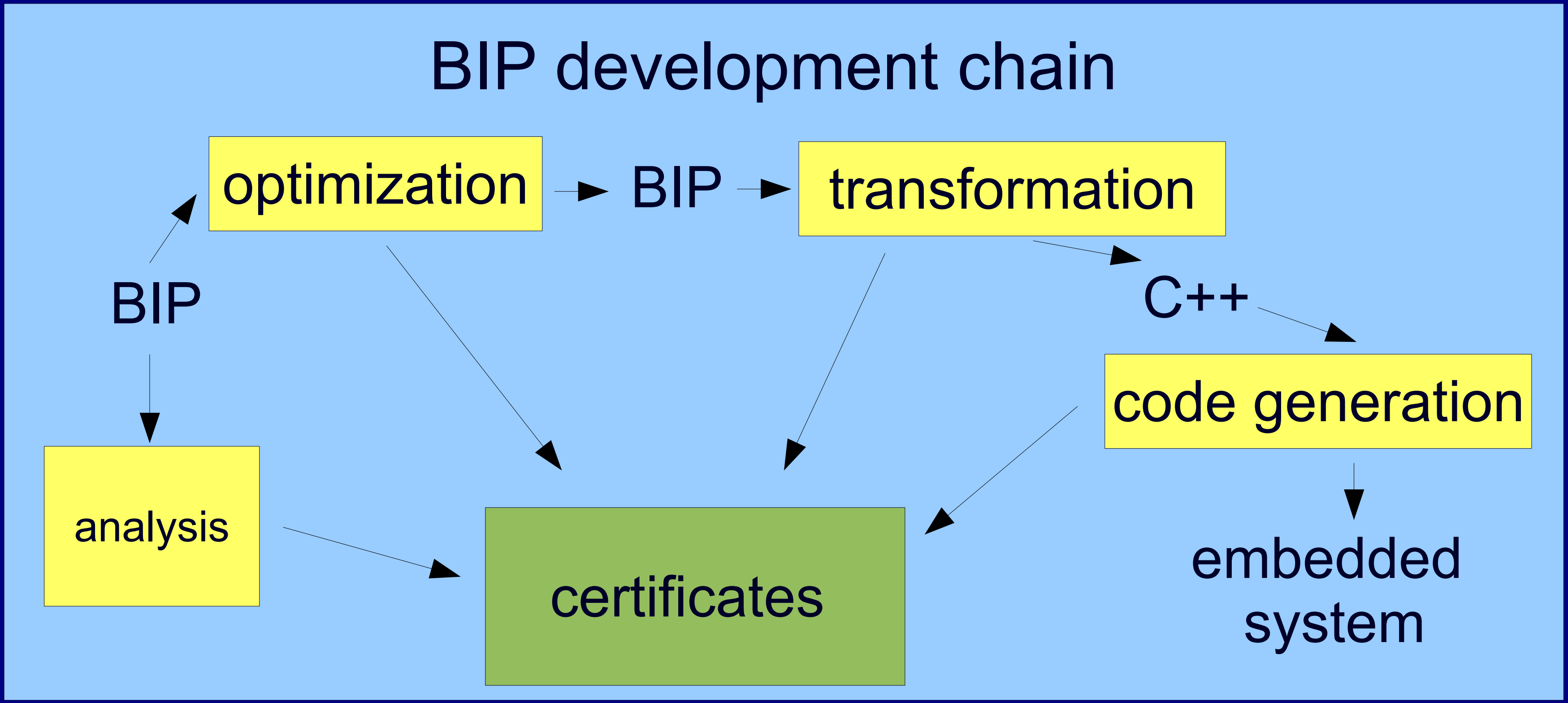
- in previous work we used them to prove code generation correct
  - properties of mappings
  - simulation of slices of source and target code
    - transformation to special computable semantics
- for BIP we are developing a checker predicate:

  invariant holds on partially specified state $\rightarrow$

  invariant holds on successor of partially specified state

Ideas for Future Work: Certificates for BIP Models

- certifying analysis results and transformations



- lift correctness results through the development chain

Further Ideas for Future Work

- semantics
  - hierarchical components
  - exploit semantic features in certificate checking
  - higher programming language guards updates + methods to reason about them explicitly
- SMT/SAT solvers for certificate checking
  - extend them to generate Coq proof terms
- combination of certificates
- ...

Related Approaches / Work

- (Foundational) Proof Carrying Code

  [Necula, Appel,...]
- Translation Validation
  - classical approach [Pnueli, Zuck, ...]
  - scheduling algorithm in Compcert [Tristan + Leroy '08]
- documenting results of verification tools
  - model checkers [Namjoshi,Cleaveland...]
  - SAT solver [Zhang + Malik '03]

Thank you for your attention!