**The joint 3rd International Workshop on CPAchecker (CPA'18) and
8th Linux Driver Verification (LDV) Workshop
September 26, 2018**

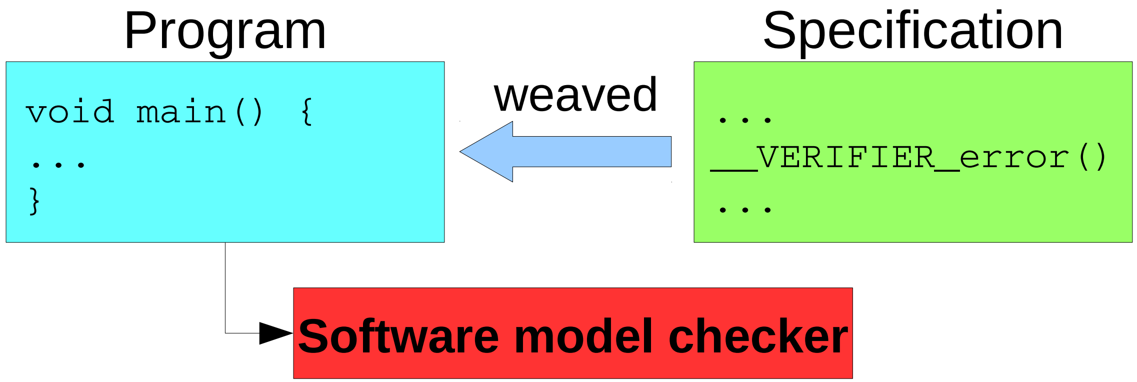
Towards Complex Specifications in Software Model Checking

Vitaly Mordan
mordan@ispras.ru

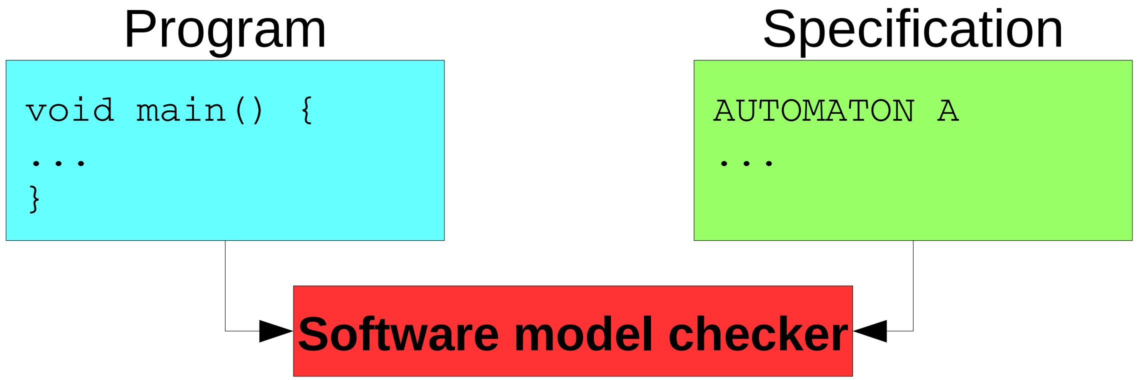
Ivannikov Institute for System Programming of the Russian Academy of Sciences

Specification in Software Model Checking

- Weaved into the source code



- Separated from the source code



Separated vs. Weaved Specifications

Weaved into the source code

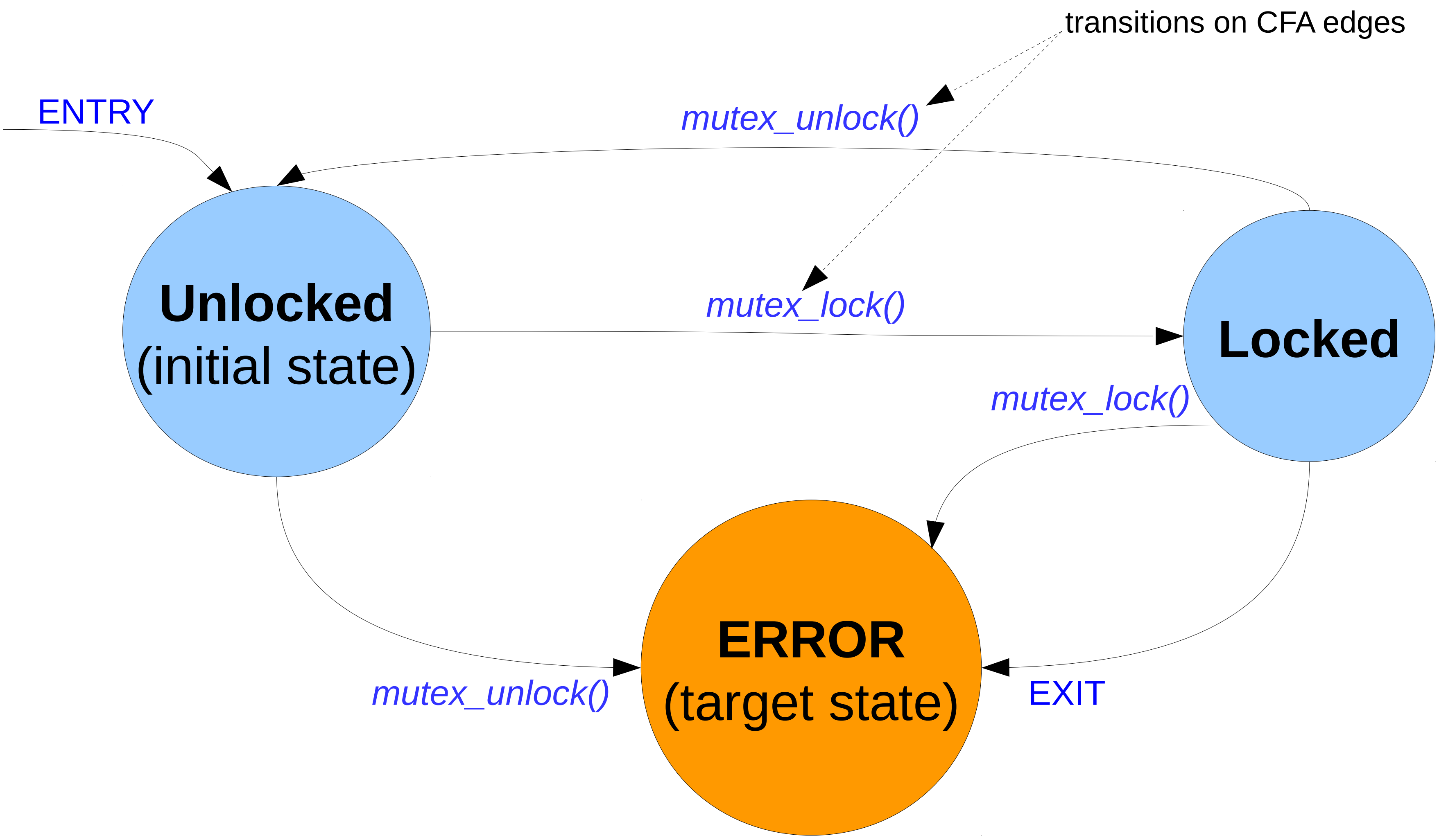
- ✓ Written in C language
- ✓ Does not require support in model checker
- ✗ Complicates the source code
- ✗ Requires additional resources

Separated from the source code

- ✗ Written in specific language
- ✗ Require additional support in model checker
- ✓ Allows to verify unchanged code
- ✓ Does not require preparation of verification tasks
- ✓ Adds new capabilities

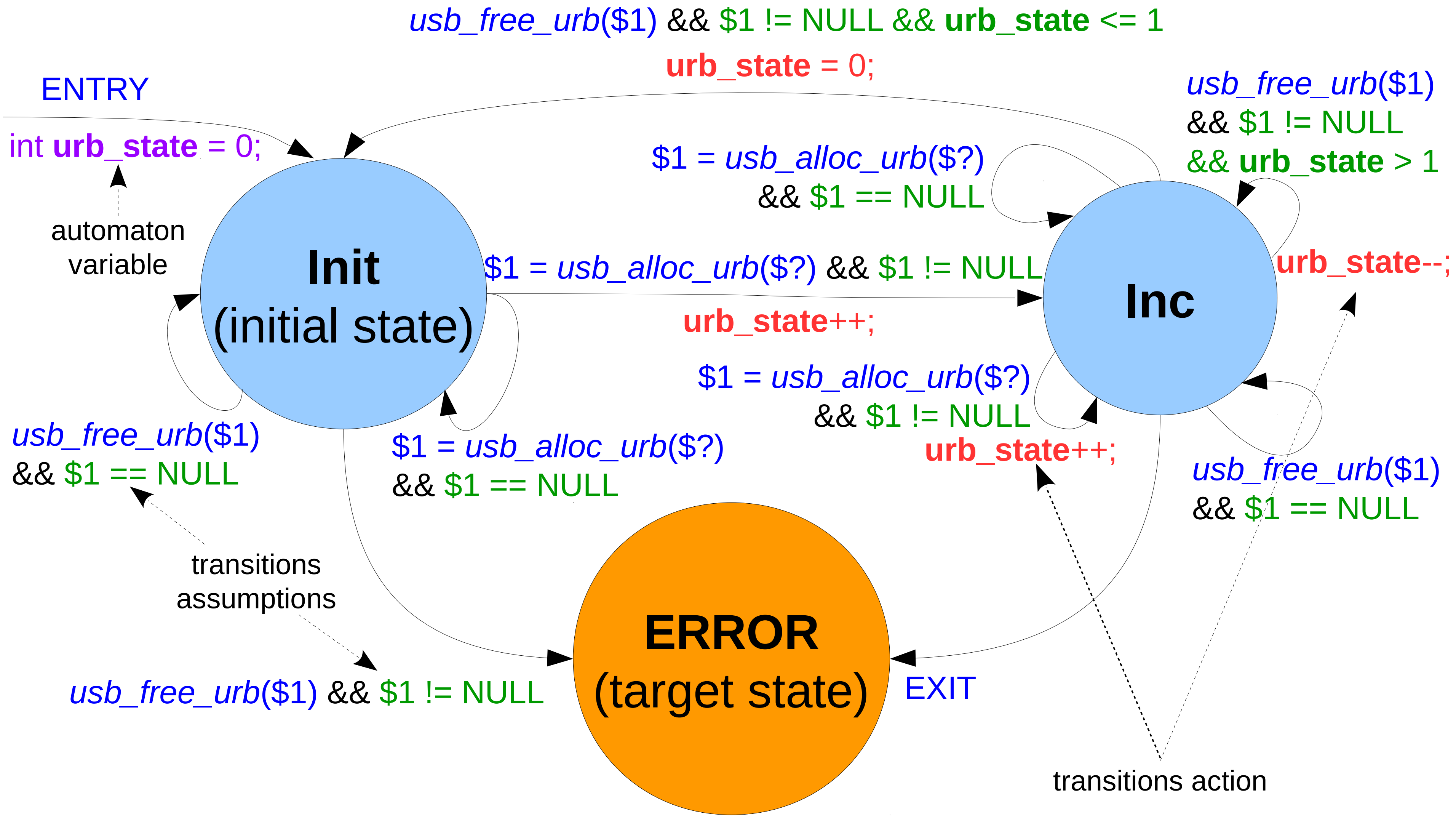
  ➡ Complex specifications

CPAchecker Specification Automata



* Based on specification of correct ussage of mutex locks *http://forge.ispras.ru/issues/1940*.
Can be presented in the CPAchecker trunk.

Automata Variables and Assumptions



Specification Automata Language*

- Automaton variables declaration (integer)
- Initial state name
- Automaton states
    - Target state → specification violation
    - Automaton transitions
        - Trigger (e.g., CFA edge)
        - Assumption (additional condition)
        - Actions (modification of automata variables)

* Full description of specification automata language can be found in *doc/SpecificationAutomata.md*.

Towards Complex Specifications

1) Interacting with other CPAs
   - ✓ Cannot be done with weaved specifications
2) Complex types of automata variables
   - Potentially more high-level data structures
   - Use internal representation for more efficient variables operations implementation
3) Specification with multiple properties
   - Increase efficiency in several times
   - Potentially can improve verification results

Interacting with Other CPAs

- Evaluation of a query

```
EVAL(location, "lineno")
```

returns line number of the analyzed CFA edge

- Checking a property

```
CHECK(location, "functionName==f")
```

returns true, if *location* CPA is inside function *f*

- Checking covered lines

```
COVERS_LINES (5 25 125)
```

returns true, if lines *5*, *25* and *125* are covered

- Modification of a CPA

```
MODIFY(ValueAnalysis, "setvalue(x:=0)")
```

sets to variable *x* value *0* in *value analysis* CPA

Example of Interacting with CPA

- Check SMG CPA for memory leaks

```
CONTROL AUTOMATON SMGCPAMEMTRACK
INITIAL STATE Init;
STATE USEFIRST Init:
  CHECK(SMGCPA, "has-leaks") -> ERROR("valid-memtrack");
END AUTOMATON
```

- Weaved specifications
  - Cannot get similar information (from SMG CPA)

Towards Complex Specifications

1) Interacting with other CPAs
   - ✓ Cannot be done with weaved specifications

2) Complex types of automata variables
   - Potentially more high-level data structures
   - Use internal representation for more efficient variables operations implementation

3) Specification with multiple properties
   - Increase efficiency in several times
   - Potentially can improve verification results

Complex Types of Automata Variables

Weaved into the source code

- ✓ Supported: program variables of any C type
- ✗ Limited support of container types (sets, maps, etc.)
  - ✗ Inefficient due to complex solver queries
  - ✗ Heuristics lead to false alarms and missed bugs

Separated from the source code

- ✗ Supported: integer automaton variables
- ✓ Required support
  - ➡ Container types
  - ➡ Pointers to program variables

Automaton Set Variables

- Represent set of some elements
  - Supported elements type: string and integer*
- Support basic set operations
  - **Add** a new element
  - **Remove** an element
  - Check if set **contains** an element
  - Check if set is **empty**

Transition actions (Add, Remove)

Transition assumptions (contains, empty)

➡ Other container types can be implemented similarly

* Available in the CPAchecker trunk from revision 28054.

Example of Specification Automaton*

```
AUTOMATON mutex_locks
LOCAL set<string> acquired_mutexes = [];     Declaration of the set
INITIAL STATE Init;
STATE Init:
  MATCH {mutex_lock($1)} ->                   Check if set contains the element
    ASSUME {$$acquired_mutexes[$1]}
    ERROR("mutex_lock:double_lock");
  MATCH {mutex_lock($1)} ->                   Add element to the set
    DO acquired_mutexes[$1]=true GOTO Init;
  MATCH {mutex_unlock($1)} ->
    ASSUME {!$$acquired_mutexes[$1]}
    ERROR("mutex_lock:double_unlock");
  MATCH {mutex_unlock($1)} ->                 Remove element from the set
    DO acquired_mutexes[$1]=false GOTO Init;
  MATCH {check_final_state($?)} ->            Check if the set is empty
    ASSUME {!$$acquired_mutexes.empty}
    ERROR("mutex_lock:locked_at_exit");
END AUTOMATON
```

* Based on specification of correct ussage of mutex locks *http://forge.ispras.ru/issues/1940*.
In comparison with previous automaton, it provides in about 10 times less false alarms.

Examples for Set Variable

Example of correct program

```
struct mutex;

void main(void) {
  struct mutex *mutex_1;
  struct mutex *mutex_2;
  mutex_lock(&mutex_1);
  mutex_lock(&mutex_2);
  mutex_unlock(&mutex_2);
  mutex_unlock(&mutex_1);
}
```

Example of incorrect program

```
struct mutex;

void main(void) {
  struct mutex *mutex_1;
  struct mutex *mutex_2;
  mutex_lock(&mutex_1);
  mutex_lock(&mutex_2);
  mutex_unlock(&mutex_1);
  mutex_unlock(&mutex_1);
}
```

Automaton Set Variables Evaluation

- LDV benchmark
  - 4124 tasks based on Linux kernel 4.1-rc1
  - 900 seconds of CPU time / 15GB of RAM
  - CPAchecker trunk, revision 28054*

Property	Specification automata with set variables				Weaved specifications with the LDV “arg_sign” heuristic**			
	Safe	Unsafe	Unknown	CPU	Safe	Unsafe	Unknown	CPU
linux:mmc	4 034	3	87	108 000	4 025	3	96	122 000
linux:mutex	3 956	44	124	148 000	3 956	42	126	166 000
linux:spinlock	4 005	20	99	120 000	3 946	19	159	208 000
Overall	11 995	67	310	376 000	11 927	64	381	496 000

+71 solved tasks

x1.3 faster

~150 000 CPU seconds per specification for its weaving

* With workaround to process set elements by the LDV “arg_sign” heuristic**.

** For expressions comparison (e.g., expression “&dev→mutex” → string “mutex_of_device”).

Pointers to Program Variables

- Mapping from the program variable to the automaton variable
- Required support
    - Evaluation in transition assumptions
- Main goal – usage inside container types
    - Keep variables values instead of their names
    - Get rid of heuristics (e.g., LDV “arg_sign”)
    - Sound and precise specifications

Motivating Examples

Example of correct program

```
struct mutex;
void f(struct mutex *m1) {
  mutex_lock(m1);
}
                m1 == &mutex_1
void g(struct mutex *m2) {
  mutex_lock(m2);
}
                m2 == &mutex_2
void main(void) {
  struct mutex mutex_1;
  struct mutex mutex_2;
  f(&mutex_1);
  g(&mutex_2);
  mutex_unlock(&mutex_1);
  mutex_unlock(&mutex_2);
}
```

Example of incorrect program

```
struct mutex;
void f(struct mutex *m) {
  mutex_lock(m);
}
                m == &mutex_1
void g(struct mutex *m) {
  mutex_unlock(m);
}
                m == &mutex_2
void main(void) {
  struct mutex mutex_1;
  struct mutex mutex_2;
  f(&mutex_1);
  g(&mutex_2);
}
```

Containers of Pointers Support

- Required support for set of pointers
  - **Remove** an element

Current set: `s=[p_1, ..., p_N]`

User interface: `DO s[p]=false`

Implementation:

```
=> ASSUME {(p == p_1)} DO s[p_1]=false
   ...
   ASSUME {(p == p_N)} DO s[p_N]=false
```

create several transitions dynamically

  - Check if set **contains** an element / **add** element

Current set: `s=[p_1, ..., p_N]`

User interface: `ASSUME {$$s[p]}`

Implementation:

```
=> ASSUME {(p == p_1) || ... || (p == p_N)}
```

not supported

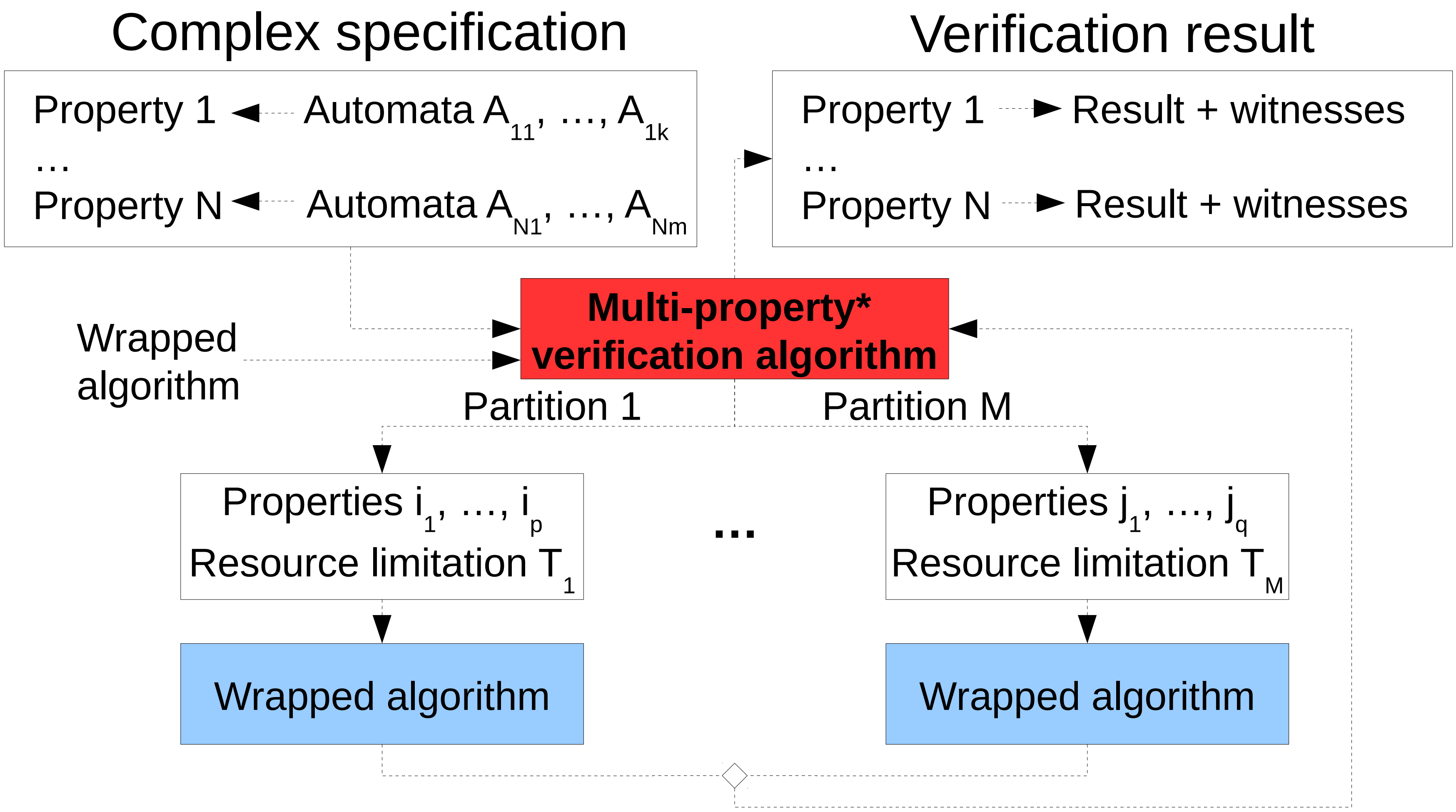
Complex Types of Automata Variables

- Previously supported
  - Integer
    - Flags and counters
- Added support
  - Container types: set<integer>, set<string>
    - ➡ Improves efficiency for complex specifications
- Planned support
  - Pointer, set<pointer>, other container types
    - ➡ Sound and precise specifications

Towards Complex Specifications

1) Interacting with other CPAs
   - ✓ Cannot be done with weaved specifications
2) Complex types of automata variables
   - Potentially more high-level data structures
   - Use internal representation for more efficient variables operations implementation
3) **Specification with multiple properties**
   - Increase efficiency in several times
   - Potentially can improve verification results

Specification with Multiple Properties



➡ Successfully used to increase efficiency in several times with the same results*

* S. Apel, D. Beyer, V. Mordan, V. Mutilin, A. Stahlbauer. *On-The-Fly Decomposition of Specifications in Software Model Checking*. FSE 2016.
Soon it will be available in the CPAchecker trunk.

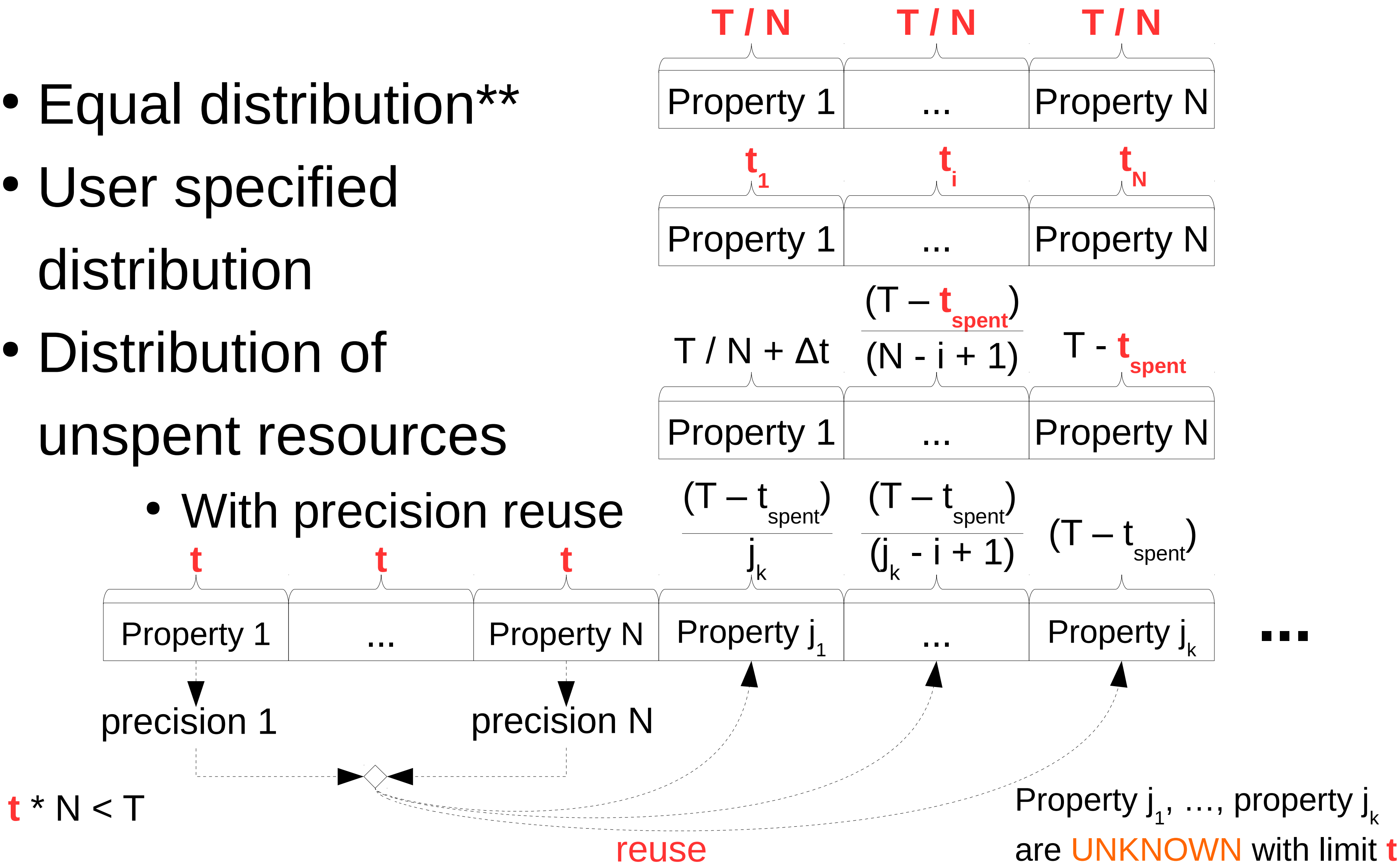
Further Development Directions

- Resource distribution algorithms
  - Focus on efficiency or effectiveness
- Partitioning algorithms
  - Conditional multi-aspect verification*
- Different wrapped algorithms
  - Memory safety, etc.

* V. Mordan, V. Mutilin. *Checking Several Requirements at once by CEGAR*. LNCS 9609.

Resource Distribution Algorithms*

- Equal distribution**
- User specified distribution
- Distribution of unspent resources
  - With precision reuse



* Assuming, that ***N*** properties and overall limitation ***T*** are given.

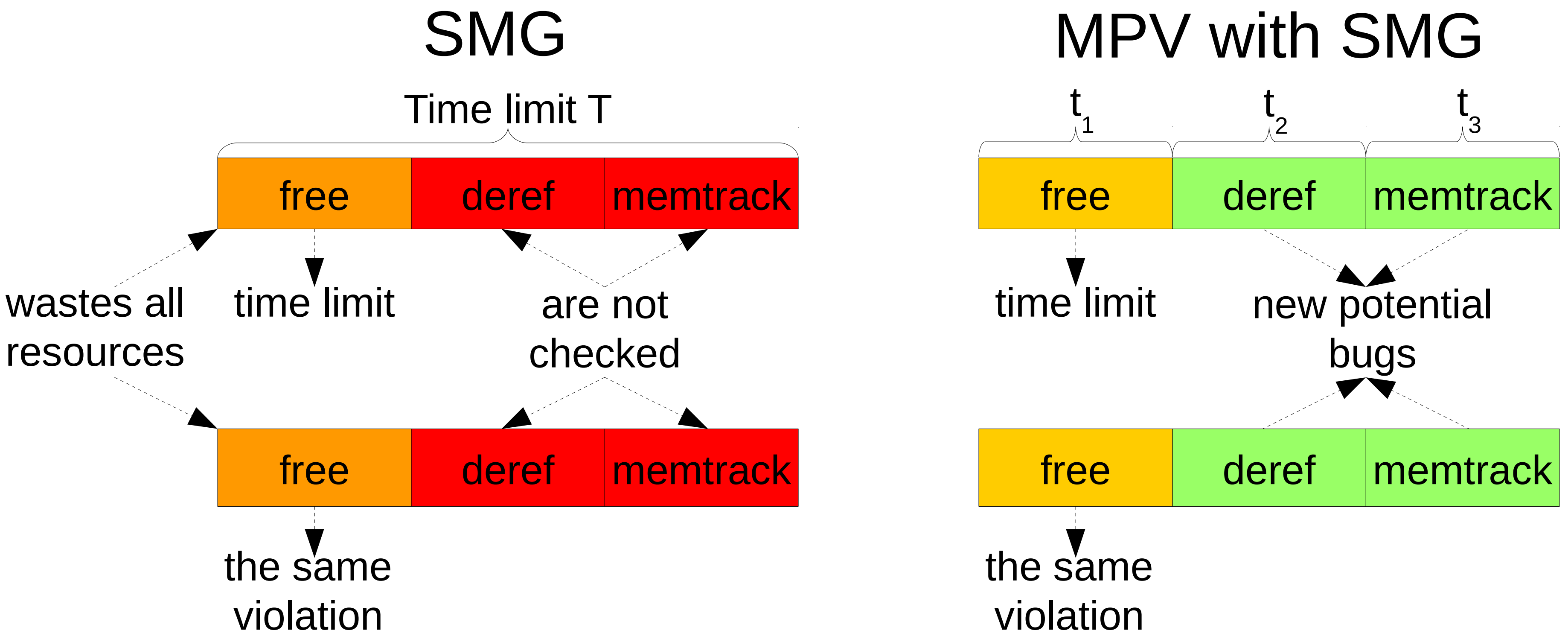
** S. Apel, D. Beyer, V. Mordan, V. Mutilin, A. Stahlbauer. *On-The-Fly Decomposition of Specifications in Software Model Checking*. FSE 2016.

Example with SMG Wrapped Algorithm

- SVCOMP memory safety specification
  - 3 property (“*free*”, “*deref*”, “*memtrack*”)
- Task
  - Find more different violations with a given resource limitation (focus on effectiveness)
- Solutions
  - Symbolic memory graph (SMG)
    - Use all resources for the whole specification
  - Multi-property verification with SMG
    - Complex specification with 3 properties
    - Distribute resources over each property

MPV with SMG Preliminary Results

- Found 40% more potential bugs than SMG
  - With distribution of unspent resources
- Required comparable amount of resources



Conclusion

- Specification automata support more complex specifications
- Complex types of automata variables
  - Container types → efficiency increase
  - Pointer to program variable → sound and precise
- Specification with multiple properties
  - Can improve verification results
  - Can use other wrapped algorithms

Future Plans

- Specification automaton development
  - Support of complex types for automata variables
- Multi-property verification development
  - New resource distribution algorithms
    - Precision reuse → improving verification results
  - New partitioning algorithms
    - Conditional multi-aspect verification
- Integration with BenchExec (SVCOMP)
  - Support specification automata as property
  - Support multi-property verification

Thank you

Vitaly Mordan
mordan@ispras.ru

Ivannikov Institute for System Programming of the Russian Academy of Sciences