Symblic Execution Model

Thi Thu Ha Doan[0000−0001−7524−4497] and Peter Thiemann[0000−0002−9000−1239]

University of Freiburg, Germany
{doanha,thiemann}@informatik.uni-freiburg.de

Abstract. **Keywords:**

1 Introduction

2 Symblic Execution Model

Let $S = (t_1, \mathrm{ty}_1) :: (t_2, \mathrm{ty}_2) :: \ldots :: [\,]$ be a stack, where elements are types paired with terms.
Let $I = i_1; i_2; \ldots; i_n$ be a sequence of instructions.
Let P be a predicate.

Definition 1. *A system state of the symbolic execution is a tuple* $ST = [I, S, S', P]$, *where* S *is the main stack and* S' *is a temporary stack. Let* $SE = \{ST_1, ST_2, \ldots, ST_n\}$ *range over sets of system states.*

Let S_{init} be the initial main stack.

$$S_{init} = (\text{Pair } \mathit{par}\ \mathit{stg}, \text{pair ty}_1\ \text{ty}_2) :: [\,]$$

where *par* and *stg* are the terms that represent the parameter and the storage. Let S_{final} be the final main stack.

$$S_{final} = (\text{Pair } \mathit{opl}\ \mathit{stg}, \text{pair (operation list) ty}_2) :: [\,]$$

where *opl* repesents a operation list

$$
\begin{array}{rl}
T, U ::= & \\
& \mid \langle \text{comparable type} \rangle \\
& \mid \text{option}\langle \text{type} \rangle \\
& \mid \text{list}\langle \text{type} \rangle \\
& \mid \text{set}\langle \text{comparable type} \rangle \\
& \mid \text{operation} \\
& \mid \text{contract}\langle \text{type} \rangle \\
& \mid \text{ticket}\langle \text{comparable type} \rangle \\
& \mid \text{pair}\langle \text{type} \rangle\langle \text{type} \rangle \\
& \mid \text{or}\langle \text{type} \rangle\langle \text{type} \rangle \\
& \mid \text{lambda}\langle \text{type} \rangle\langle \text{type} \rangle \\
& \mid \text{map}\langle \text{comparable type} \rangle\langle \text{type} \rangle \\
& \mid \text{big-map}\langle \text{comparable type} \rangle\langle \text{type} \rangle \\
& \mid \text{bls12-381-g1} \\
& \mid \text{bls12-381-g2} \\
& \mid \text{bls12-381-fr} \\
& \mid \text{sapling-transaction}\langle \text{natural number constant} \rangle \\
& \mid \text{sapling-state}\langle \text{natural number constant} \rangle \\
& \mid \text{chest} \\
& \mid \text{chest-key} \\
\langle \text{comparable type} \rangle ::= & \\
& \mid \text{unit} \\
& \mid \text{never} \\
& \mid \text{bool} \\
& \mid \text{int} \\
& \mid \text{nat} \\
& \mid \text{string} \\
& \mid \text{chain-id} \\
& \mid \text{bytes} \\
& \mid \text{mutez} \\
& \mid \text{key-hash} \\
& \mid \text{key} \\
& \mid \text{signature} \\
& \mid \text{timestamp} \\
& \mid \text{address} \\
& \mid \text{tx-rollup-l2-address} \\
& \mid \text{option}\langle \text{comparable type} \rangle \\
& \mid \text{or}\langle \text{comparable type} \rangle\langle \text{comparable type} \rangle \\
& \mid \text{pair}\langle \text{comparable type} \rangle\langle \text{comparable type} \rangle \ldots
\end{array}
$$

Fig. 1. Types

3 Rules

The set of instructions is divided into two groups I and I'. Given an instruction i, i' is a copy version of i, where i operates on the main stack S and i' operates only on the temporary stack S'.

The rule semantic is defined by several kinds of transitions:

1. $\longrightarrow_E$ single-step evaluation of an expression in a system state,
2. $\longrightarrow_S$ internal transitions of a system state,
   PT: this relation should be non-deterministic as shown for instructions IF, IF-LEFT, and LOOP; this approach enables us to simplify the rule for DIP
3. $\longrightarrow$ symbolic system transitions.

System rules

INVALID-PRE

$$\frac{\neg\, P}{\{[I, S, S', P]\} \cup SE \longrightarrow SE}$$

Instruction rules

Control structures

EXEC

$$\frac{[I_1, (s_1, \mathrm{ty}_1) :: [\,], [\,], Q] \longrightarrow_S^* [[\,], (s_1', \mathrm{ty}_2) :: [\,], [\,], Q']}{[(\mathrm{EXEC}; I), (\{I_1\}, \mathrm{ty}_1 \;\to\; \mathrm{ty}_2) :: (s_1, \mathrm{ty}_1) :: S, S', P \wedge Q] \longrightarrow_S [I, (s_1', \mathrm{ty}_2) :: S, S', P \wedge Q']}$$

APPLY

$$\frac{}{[(\mathrm{APPLY}; I), (s_1, \mathrm{ty}_1) ::\; (\{I_1\}, \mathrm{lambda}\ (\mathrm{pair}\ \mathrm{ty}_1\ \mathrm{ty}_2)\ \mathrm{ty}_3) ::\; S, S', P] \longrightarrow_S [I, (\{\mathrm{PUSH}\ \mathrm{ty}_1\ s_1; \mathrm{PAIR}; I_1\}, \mathrm{lambda}\ \mathrm{ty}_2\ \mathrm{ty}_3) ::\; S, S', P]}$$

LAMBDA

$$\frac{}{[(\mathrm{LAMBDA}\ \mathrm{ty}_1\ \mathrm{ty}_2\ \{I_1\}; I), S, S', P] \longrightarrow_S [I, (\{I_1\}, \mathrm{lambda}\ \mathrm{ty}_1 \;\to\; \mathrm{ty}_2) ::\; S, S', P]}$$

IF-TRUE

$$\frac{}{[(\mathrm{IF}\ I_1\ I_2; I), (s_1, \mathrm{bool}) ::\; S, S', P] \longrightarrow_S [I_1, S, S', P \;\wedge\; s_1]}$$

IF-FALSE

$$\frac{}{[(\mathrm{IF}\ I_1\ I_2; I), (s_1, \mathrm{bool}) ::\; S, S', P] \longrightarrow_S [I_2, S, S', P \;\wedge\; \neg\, s_1]}$$

IF-LEFT-LEFT

$$\frac{}{\begin{array}{c}[(\text{IF-LEFT } I_1\ I_2; I), (s_1, \text{or ty}_1\ \text{ty}_2) :: S, S', P] \longrightarrow_S \\ [I_1, (x, \text{ty}_1) ::\ S, S', P \wedge (s_1 = \text{Left } x)]\end{array}}$$

IF-LEFT-RIGHT

$$\frac{}{\begin{array}{c}[(\text{IF-LEFT } I_1\ I_2; I), (s_1, \text{or ty}_1\ \text{ty}_2) :: S, S', P] \longrightarrow_S \\ [I_2, (x, \text{ty}_2) ::\ S, S', P \wedge (s_1 = \text{Right } x))]\end{array}}$$

IF-CONS-EMPTY

$$\frac{}{\begin{array}{c}[(\text{IF-CONS } I_1\ I_2\ ; I), (s_1, \text{list ty}) ::\ S, S', P] \longrightarrow_S \\ [I_2\ ; I, S, S', P\ \wedge\ (s_1 = \{\})]\end{array}}$$

IF-CONS-NONEMPTY

$$\frac{}{\begin{array}{c}[(\text{IF-CONS } I_1\ I_2\ ; I), (s_1, \text{list ty}) ::\ S, S', P], SE\ \longrightarrow_S \\ [I_1, (hd, \text{ty}) ::\ (\{< tl >\}, \text{list ty}) ::\ S, S', P\ \wedge\ (s_1 = \{hd\ ; \langle tl \rangle\})]\end{array}}$$

IF-NONE-NONE

$$\frac{}{\begin{array}{c}[(\text{IF-NONE } I_1\ I_2\ ; I), (s_1, \text{option ty}) ::\ S, S', P], SE\ \longrightarrow_S \\ [I_1\ ; I, S, S', P\ \wedge\ (s_1 = \text{None})]\end{array}}$$

IF-NONE-SOME

$$\frac{}{\begin{array}{c}[(\text{IF-NONE } I_1\ I_2\ ; I), (s_1, \text{option ty}) ::\ S, S', P], SE\ \longrightarrow_S \\ [I_2, (x, \text{ty}) ::\ S, S', P\ \wedge\ (s_1 = \text{Some } x)]\end{array}}$$

LOOP-TRUE

$$\frac{}{[(\text{LOOP } I_1; I), (s_1, \text{bool}) ::\ S, S', P] \longrightarrow_S [(I_1; \text{LOOP } I_1; I), S, S', P \wedge s_1]}$$

LOOP-FALSE

$$\frac{}{[(\text{LOOP } I_1; I), (s_1, \text{bool}) ::\ S, S', P] \longrightarrow_S [I, S, S', P \wedge (\neg s_1)]}$$

ITER-EMPTY

$$\frac{}{[(\text{ITER } I_1; I), (s_1, \text{list ty}) ::\ S, S', P] \longrightarrow_S [I, S, S', P\ \wedge\ (s_1 = \{\})]}$$

ITER-NONEMPTY

$$\frac{}{\begin{array}{c}[(\text{ITER } I_1; I), (s_1, \text{list ty}) ::\ S, S', P] \longrightarrow_S \\ [(\text{ITER' } I_1; I), S, (s_1, \text{list ty}) ::\ S', P\ \wedge\ \neg\ (s_1 = \{\})]\end{array}}$$

ITER-NONEMPTY

$$\frac{[I_1, (hd, \text{ty}) :: S, [\,], Q] \longrightarrow^*_S [[\,], S_1, [\,], Q']}{\begin{array}{c}[(\text{ITER } I_1; I), (s_1, \text{list ty}) ::\ S, S', P \ \wedge\ Q] \longrightarrow_S \\ [(\text{ITER } I_1; I), (\langle tl \rangle, \text{list ty}) ::\ S_1, S', P \ \wedge\ (s_1 = \{hd\,; \langle tl \rangle\}) \wedge\ Q']\end{array}}$$

ITER'-EMPTY

$$\frac{}{[(\text{ITER' } I_1; I), S, (s_1, \text{list ty}) ::\ S', P] \longrightarrow_S [I, S, S', P \ \wedge\ (s_1 = \{\})]}$$

ITER'-NONEMPTY

$$\frac{}{\begin{array}{c}[(\text{ITER' } I_1; I), S, (s_1, \text{list ty}) ::\ S', P] \longrightarrow_S \\ [(I_1; \text{ITER' } I_1; I), (hd, \text{ty}) ::\ S, (\{< tl >\}, \text{list ty}) ::\ S', P \ \wedge\ (s_1 = \{hd\,; \langle tl \rangle\})]\end{array}}$$

Stack Manipulation

DIG

$$\frac{\text{len}(A) = n}{[(\text{DIG } n; I), A \ @\ (s_1, \text{ty}) ::\ B, S', P] \longrightarrow_S [I, (s_1, \text{ty}) ::\ A \ @\ B, S', P]}$$

DIP

$$\frac{[I_1, S, [\,], Q] \longrightarrow^*_S [[\,], S_1, [\,], Q']}{[(\text{DIP } I_1; I), (s_1, \text{ty}) :: S, S', P \wedge Q] \longrightarrow_S [I, (s_1, \text{ty}) ::\ S_1, S', P \wedge Q']}$$

DIP N

$$\frac{\text{len}(A) = n \qquad [I_1, B, [\,], Q] \longrightarrow^*_S [[\,], B_1, [\,], Q']}{[(\text{DIP } n\ I_1; I), A \ @\ B, S', P \wedge Q] \longrightarrow_S [(I), A \ @\ B_1, S', P \wedge Q']}$$

PUSH

$$\frac{}{[(\text{PUSH ty } x\ ; I), S, S', P] \longrightarrow_S \ [I, (x, \text{ty}) ::\ S, S', P]}$$

Arithmetic operations

ADD

$$\frac{}{\begin{array}{c}[(\text{ADD }; I), (s_1, \text{nat}) ::\ (s_2, \text{nat}) ::\ S, S', P] \longrightarrow_S \\ [I, (x, \text{nat}) ::\ S, S', P \wedge\ (x = s_1 + s_2)]\end{array}}$$

ABS

$$\frac{}{\begin{array}{c}[(\text{ABS }; I), (s_1, \text{int}) :: S, S', P] \longrightarrow_S \\ [I, (x, \text{nat}) :: S, S', P \wedge (s_1 \geq 0 \Rightarrow x = s_1) \wedge (s_1 < 0 \Rightarrow x = -s_1)]\end{array}}$$

PT: I think COMPARE can be simplified as follows

$$\text{COMPARE-NAT}\quad \frac{}{\begin{array}{c}[(\text{COMPARE}; I), (s_1, \text{nat}) :: (s_2, \text{nat}) :: S, S', P] \longrightarrow \\ [I, (x, \text{int}) :: S, S', P \wedge\ (s_1 > s_2 \Leftrightarrow x = 1) \wedge\ (s_1 = s_2 \Leftrightarrow x = 0) \\ \wedge\ (s_1 < s_2 \Leftrightarrow x = -1)]\end{array}}$$

$$\text{COMPARE-OPTION-SOME}\quad \frac{[\text{COMPARE}, (x, \text{ty}) :: (y, \text{ty}) ::\ [\,], [\,], Q] \longrightarrow_S^* [[\,], (a, \text{int}) ::\ [\,], [\,], Q']}{\begin{array}{c}[(\text{COMPARE}; I), (s_1, \text{option ty}) :: (s_2, \text{option ty}) :: S, S', P \wedge Q] \longrightarrow \\ [I, (a, \text{int}) :: S, S', P \wedge (s_1 = \text{Some } x) \wedge (s_2 = \text{Some } y) \wedge Q']\end{array}}$$

$$\text{COMPARE-OPTION-NONE}\quad \frac{}{\begin{array}{c}[(\text{COMPARE}; I), (s_1, \text{option ty}) :: (s_2, \text{option ty}) :: S, S', P] \longrightarrow \\ [I, (1, \text{int}) :: S, S', P \wedge (s_1 = \text{Some } x) \wedge (s_2 = \text{None})]\end{array}}$$

Boolean operations

$$\text{XOR}\quad \frac{}{\begin{array}{c}[(\text{XOR} ; I), (s_1, \text{bool}) ::\ (s_2, \text{bool}) ::\ S, S', P] \longrightarrow_S \\ [I, (x, \text{bool}) ::\ S, S', P \wedge\ (x = s_1 \text{ xor } s_2)]\end{array}}$$

Crytographic oprerations

$$\text{HASH-KEY}\quad \frac{}{\begin{array}{c}[(\text{HASH-KEY} ; I), (s_1, \text{byte}) ::\ S, S', P] \longrightarrow_S \\ [I, (x, \text{byte}) ::\ S, S', P \wedge\ (x\ = \text{hash-key}(s_1))]\end{array}}$$

Blockchain operations

$$\text{AMOUNT}\quad \frac{}{[(\text{AMOUNT} ; I), S, S', P] \longrightarrow_S [I, (\text{amount}, \text{mutez}) ::\ S, S', P}$$

$$\text{CONTRACT TY - SOME}\quad \frac{}{\begin{array}{c}[(\text{CONTRACT ty}; I), (s_1, \text{address}) ::\ S, S', P] \longrightarrow \\ [I, (\text{Some } x, \text{option (contract ty)}) ::\ S, S', \\ P \wedge\ (\text{get-contract-type}(s_1, \text{ty}) = \text{Some } x)]\end{array}}$$

$$\text{CONTRACT TY - NONE}\quad \frac{}{\begin{array}{c}[(\text{CONTRACT ty}; I), (s_1, \text{address}) ::\ S, S', P] \longrightarrow \\ [I, \text{None} ::\ S, S', P \wedge\ (\text{get-contract-type}(s_1, \text{ty}) = \text{None}]\end{array}}$$

Operations on data structures

$$\frac{\text{CAR}}{\begin{array}{c}[(\text{CAR}\ ; I), (s_1, \text{pair ty}_1\ \text{ty}_2) ::\ S, S', P] \longrightarrow_S \\ [I, (x, \text{ty}_1) ::\ S, S', P\ \wedge\ (s_1 = \text{Pair}\ x\ y)]\end{array}}$$

$$\frac{\text{CONCAT}}{\begin{array}{c}[(\text{CONCAT}\ ; I), (s_1, \text{list string}) ::\ S, S', P] \longrightarrow_S \\ [(\text{CONCAT'}\ ; I), (\text{""}, \text{string}) ::\ S, (s_1, \text{list string}) ::\ S', P]\end{array}}$$

$$\frac{\text{CONCAT'}}{[(\text{CONCAT'}\ ; I), S, (s_2, \text{string}) ::\ S', P] \longrightarrow_S [I, S, S', P\ \wedge\ (s_2 = \{\})]}$$

$$\frac{\text{CONCAT'}}{\begin{array}{c}[(\text{CONCAT'}\ ; I), (s_1, \text{string}) ::\ S, (s_2, \text{list string}) ::\ S', P] \longrightarrow_S \\ [(\text{CONCAT'}\ ; I), (s_1 \hat{}\ hd\ ::\ S, \text{string}), \\ (\{< tl >\}, \text{list string}) ::\ S', P\ \wedge\ (s_2 = \{hd\ ; < tl >\})]\end{array}}$$

$$\frac{\text{MEM-EMPTY}}{\begin{array}{c}[(\text{MEM}\ ; I), (s_1, \text{ty}_1) ::\ (s_2, \text{map ty}_1\ \text{ty}_2) ::\ S, S', P] \longrightarrow_S \\ [I, \text{False} ::\ S, S', P\ \wedge\ (s_2 = \{\})]\end{array}}$$

MEM-NONEMPTY

$$\frac{[\text{COMPARE}, (s_1, \text{ty}_1) ::\ (k, \text{ty}_1) ::\ [\,], [\,], Q] \longrightarrow_S^* [[\,], (b, \text{int}) ::\ [\,], [\,], Q']}{\begin{array}{c}[(\text{MEM}\ ; I), (s_1, \text{ty}_1) ::\ (s_2, \text{map ty}_1\ \text{ty}_2) ::\ S, S', P\ \wedge\ Q] \longrightarrow_S \\ [\text{MEM}; I, s_1\ ::\ \{< m >\} ::\ S, S', \\ P\ \wedge\ Q' \wedge\ (s_2 = \{\mathit{Elt}\ k\ v\ ; < m >\}) \wedge\ (b = 1)]\end{array}}$$

MEM-NONEMPTY

$$\frac{[\text{COMPARE}, (s_1, \text{ty}_1) ::\ (k, \text{ty}_1) ::\ [\,], [\,], Q] \longrightarrow_S^* [[\,], (b, \text{bool}) ::\ [\,], [\,], Q']}{\begin{array}{c}[(\text{MEM}\ ; I), (s_1, \text{ty}_1) ::\ (s_2, \text{map ty}_1\ \text{ty}_2) ::\ S, S', P\ \wedge\ Q] \longrightarrow_S \\ [I, \text{True}\ ::\ S, S', P\ \wedge\ Q' \wedge\ (s_2 = \{\mathit{Elt}\ k\ v\ ; < m >\}) \wedge\ (b = 0)]\end{array}}$$

MEM-NONEMPTY

$$\frac{[\text{COMPARE}, (s_1, \text{ty}_1) ::\ (k, \text{ty}_1) ::\ [\,], [\,], Q] \longrightarrow_S^* [[\,], (b, \text{bool}) ::\ [\,], [\,], Q']}{\begin{array}{c}[(\text{MEM}\ ; I), (s_1, \text{ty}_1) ::\ (s_2, \text{map ty}_1\ \text{ty}_2) ::\ S, S', P\ \wedge\ Q] \longrightarrow_S \\ [I, \text{False}\ ::\ S, S', P\ \wedge\ Q' \wedge\ (s_2 = \{\mathit{Elt}\ k\ v\ ; < m >\}) \wedge\ (b = -1)]\end{array}}$$

MAP-EMPTY

$$\frac{}{\begin{array}{c}[(\text{MAP } I_1; I), (s_1, \text{list ty}) ::\ S, S', P] \longrightarrow_S \\ [I, (s_1, \text{list ty}) ::\ S, S', P\ \wedge\ (s_1 = \{\})]\end{array}}$$

MAP-NONEMPTY

$$\frac{}{\begin{array}{c}[(\text{MAP } I_1; I), (s_1, \text{list ty}) ::\ S, S', P] \longrightarrow_S \\ [(\text{MAP' } I_1; I), (\{\}, \text{list ty}) ::\ S, (s_1, \text{list ty}) ::\ S', P\ \wedge\ \neg(s_1 = \{\})]\end{array}}$$

MAP'-EMPTY

$$\frac{}{[(\text{MAP' } I_1; I), S, (s_2, \text{list ty}) ::\ S', P] \longrightarrow_S [I, S, S', P\ \wedge\ (s_2 = \{\})]}$$

MAP'-NONEMPTY

$$\frac{[I_1, (hd, \text{ty}) ::\ S, [\,], Q] \longrightarrow_S^* [[\,], (hd', \text{ty}) ::\ S, [\,], Q']}{\begin{array}{c}[(\text{MAP' } I_1; I), (s_1, \text{list ty}) ::\ S, (s_2, \text{list ty}) ::\ S', P\ \wedge\ Q] \longrightarrow_S \\ [(\text{MAP' } I_1; I), (s_1@\ \{hd'\}, \text{list ty}) ::\ S, (\{< tl >\}, \text{list ty}) ::\ S', \\ P\ \wedge\ Q' \wedge\ (s_2 = \{hd; < tl >\})]\end{array}}$$

UPDATE-EMPTY-TRUE

$$\frac{}{\begin{array}{c}[(\text{UPDATE }; I), (x, \text{ty}) ::\ (b, \text{bool}) ::\ (\{\}, \text{list ty}) ::\ S, S', P]\ \longrightarrow \\ [I, (\{x\}, \text{list ty}) ::\ S, S', P\ \wedge\ (b = \text{True})]\end{array}}$$

UPDATE-EMPTY-FALSE

$$\frac{}{\begin{array}{c}[(\text{UPDATE }; I), (x, \text{ty}) ::\ (b, \text{bool}) ::\ (\{\}, \text{list ty}) ::\ S, S', P]\ \longrightarrow \\ [I, (\{\}, \text{list ty})\ ::\ S, S', P\ \wedge\ (b = \text{False})]\end{array}}$$

UPDATE-NONEMPTY-TRUE-0

$$\frac{[\text{COMPARE}, (x, \text{ty}) ::\ (hd, \text{ty}) ::\ [\,], [\,], Q] \longrightarrow_S^* [[\,], (a, \text{int}) ::\ [\,], [\,], Q']}{\begin{array}{c}[(\text{UPDATE }; I), (x, \text{ty}) ::\ (b, \text{bool}) ::\ (s_1, \text{list ty}) ::\ S, S', P\ \wedge\ Q] \longrightarrow \\ [I, (s_1, \text{list ty}) ::\ S, S', \\ P\ \wedge\ Q' \wedge\ (s_1 = \{hd; < tl >\}) \wedge\ (b = \text{True}) \wedge\ (a = 0)]\end{array}}$$

UPDATE-NONEMPTY-FALSE-0

$$\frac{[\text{COMPARE}, (x, \text{ty}) ::\ (hd, \text{ty}) ::\ [\,], [\,], Q] \longrightarrow_S^* [[\,], (a, \text{int}) ::\ [\,], [\,], Q']}{\begin{array}{c}[(\text{UPDATE }; I), (x, \text{ty}) ::\ (b, \text{bool}) ::\ (s_1, \text{list ty}) ::\ S, S', P\ \wedge\ Q] \longrightarrow \\ [I, (\{< tl >\}, \text{list ty}) ::\ S, S', \\ P\ \wedge\ Q' \wedge\ (s_1 = \{hd; < tl >\}) \wedge\ (b = \text{False}) \wedge\ (a = 0)]\end{array}}$$

UPDATE-NONEMPTY-TRUE- (-1)

$$\frac{[\text{COMPARE}, (x, \text{ty}) ::\ (hd, \text{ty}) ::\ [\,], [\,], Q] \longrightarrow_S^* [[\,], (a, \text{int}) ::\ [\,], [\,], Q']}{\begin{gathered}[(\text{UPDATE}\ ; I), (x, \text{ty}) ::\ (b, \text{bool}) ::\ (s_1, \text{list ty}) ::\ S, S', P\ \wedge\ Q] \longrightarrow \\ [I, (\{x; hd; <\ tl >\}, \text{list ty}) ::\ S, S', \\ P\ \wedge\ Q' \wedge\ (s_1 = \{hd; <\ tl >\}) \wedge\ (b = \text{True}) \wedge\ (a = -\,1)]\end{gathered}}$$

UPDATE-NONEMPTY-FALSE- (-1)

$$\frac{[\text{COMPARE}, (x, \text{ty}) ::\ (hd, \text{ty}) ::\ [\,], [\,], Q] \longrightarrow_S^* [[\,], (a, \text{int}) ::\ [\,], [\,], Q']}{\begin{gathered}[(\text{UPDATE}\ ; I), (x, \text{ty}) ::\ (b, \text{bool}) ::\ (s_1, \text{list ty}) ::\ S, S', P\ \wedge\ Q] \longrightarrow \\ [I, (\{hd; <\ tl >\}, \text{list ty}) ::\ S, S', \\ P\ \wedge\ Q' \wedge\ (s_1 = \{hd; <\ tl >\}) \wedge\ (b = \text{False}) \wedge\ (a = -\,1)]\end{gathered}}$$

UPDATE-NONEMPTY-1

$$\frac{[\text{COMPARE}, (x, \text{ty}) ::\ (hd, \text{ty}) ::\ [\,], [\,], Q] \longrightarrow_S^* [[\,], (a, \text{int}) ::\ [\,], [\,], Q']}{\begin{gathered}[(\text{UPDATE}\ ; I), (x, \text{ty}) ::\ (b, \text{bool}) ::\ (s_1, \text{list ty}) ::\ S, S', P\ \wedge\ Q] \longrightarrow \\ [(\text{UPDATE'}\ ; I); (\{hd\}, \text{list ty}) ::\ S, (\{<\ tl >\}, \text{list ty}) ::\ S', \\ P\ \wedge\ Q' \wedge\ (s_1 = \{hd; <\ tl >\}) \wedge\ (a = 1)]\end{gathered}}$$

UPDATE´-EMPTY-TRUE

$$\frac{}{\begin{gathered}[(\text{UPDATE'}\ ; I), (l, \text{list ty}) ::\ S, (x, \text{ty}) ::\ (b, \text{bool}) ::\ (\{\}, \text{list ty}) ::\ S', P]\ \longrightarrow \\ [I, (l\ @\ \{x\}, \text{list ty}) ::\ S, S', P\ \wedge\ (b = \text{True})]\end{gathered}}$$

UPDATE´-EMPTY-FALSE

$$\frac{}{\begin{gathered}[(\text{UPDATE'}\ ; I), (l, \text{list ty}) ::\ S, (x, \text{ty}) ::\ (b, \text{bool}) ::\ (\{\}, \text{list ty}) ::\ S', P]\ \longrightarrow \\ [I, (l, \text{list ty}) ::\ S, S', P\ \wedge\ (b = \text{False})]\end{gathered}}$$

UPDATE´-NONEMPTY-TRUE-0

$$\frac{[\text{COMPARE}, (x, \text{ty}) ::\ (hd, \text{ty}) ::\ [\,], [\,], Q] \longrightarrow_S^* [[\,], (a, \text{int}) ::\ [\,], [\,], Q']}{\begin{gathered}[(\text{UPDATE'}\ ; I), (l, \text{list ty}) ::\ S, (x, \text{ty}) ::\ (b, \text{bool}) ::\ (s_1, \text{list ty}) ::\ S', P\ \wedge\ Q] \\ \longrightarrow\ [I, (l\ @\ s_1, \text{list ty}) ::\ S, S', \\ P\ \wedge\ Q' \wedge\ (s_1 = \{hd; <\ tl >\}) \wedge\ (b = \text{True}) \wedge\ (a = 0)]\end{gathered}}$$

UPDATE´-NONEMPTY-FALSE-0

$$\frac{[\text{COMPARE}, (x, \text{ty}) ::\ (hd, \text{ty}) ::\ [\,], [\,], Q] \longrightarrow_S^* [[\,], (a, \text{int}) ::\ [\,], [\,], Q']}{\begin{gathered}[(\text{UPDATE'}\ ; I), (l, \text{list ty}) ::\ S, (x, \text{ty}) ::\ (b, \text{bool}) ::\ (s_1, \text{list ty}) ::\ S', P\ \wedge\ Q] \\ \longrightarrow\ [I, (l\ @\ \{<\ tl >\}, \text{list ty}) ::\ S, S', \\ P\ \wedge\ Q' \wedge\ (s_1 = \{hd; <\ tl >\}) \wedge\ (b = \text{False}) \wedge\ (a = 0)]\end{gathered}}$$

UPDATE´-NONEMPTY-1

$$\frac{[\text{COMPARE}, (x, \text{ty}) :: (hd, \text{ty}) :: [\,], [\,], Q] \longrightarrow^*_S [[\,], (a, \text{int}) :: [\,], [\,], Q']}{\begin{array}{c}[(\text{UPDATE'}\ ; I), (l, \text{list ty}) :: S, (x, \text{ty}) :: (b, \text{bool}) :: (s_1, \text{list ty}) :: S', P \wedge Q] \\ \longrightarrow [(\text{UPDATE'}\ ; I), (l \ @ \ \{hd\}, \text{list ty}) :: S, \\ (x, \text{ty}) :: (b, \text{bool}) :: (\{< tl >\}, \text{list ty}) :: S', \\ P \wedge Q' \wedge (s_1 = \{hd; < tl >\}) \wedge (a = 1)]\end{array}}$$

UPDATE´-NONEMPTY-TRUE–1

$$\frac{[\text{COMPARE}, (x, \text{ty}) :: (hd, \text{ty}) :: [\,], [\,], Q] \longrightarrow^*_S [[\,], (a, \text{int}) :: [\,], [\,], Q']}{\begin{array}{c}[(\text{UPDATE'}; I), (l, \text{list ty}) :: S, (x, \text{ty}) :: (b, \text{bool}) :: (s_1, \text{list ty}) :: S', P \wedge Q] \\ \longrightarrow [I, (l \ @ \ \{b; hd; < tl >\}, \text{list ty}) :: S, S', \\ P \wedge Q' \wedge (s_1 = \{hd; < tl >\}) \wedge (b = \text{True}) \wedge (a = -1)]\end{array}}$$

UPDATE´-NONEMPTY-FALSE–1

$$\frac{[\text{COMPARE}, (x, \text{ty}) :: (hd, \text{ty}) :: [\,], [\,], Q] \longrightarrow^*_S [[\,], (a, \text{int}) :: [\,], [\,], Q']}{\begin{array}{c}[(\text{UPDATE'}; I), (l, \text{list ty}) :: S, (x, \text{ty}) :: (b, \text{bool}) :: (s_1, \text{list ty}) :: S', P \wedge Q] \\ \longrightarrow [I, (l \ @ \ s_1, \text{list ty}) :: S, S', \\ P \wedge Q' \wedge (s_1 = \{hd; < tl >\}) \wedge (b = \text{False}) \wedge (a = -1)]\end{array}}$$

Operation on tickets

FAILWITH

FAILWITH

$$\frac{}{[(\text{FAILWITH}; I), S, S', P] \longrightarrow_S [\emptyset, [\,], [\,], P]}$$

4 Dealing with Loop in the Symbolic Execution

In Michelson, there are several loop instructions that execute a loop over the data structure, such as ITER, MAP, MEMBER, SIZE and UPDATE on list, set, map or on the stack, such as LOOP and LOOP-LEFT.

Let us consider the ITER instruction (for list), which applies to a stack S, where the top element has type (ty list) and the rest of the stack has type A, returns the result stack of type A. The ITER instruction loops on the datastructure list. Typing

$$\frac{\Gamma \vdash I : \text{ty} : \text{A} \ \rightarrow \ \text{A}}{\Gamma \vdash \text{ITER } I : \text{list ty} : \text{A} \ \rightarrow \ \text{A}}$$

Semantic

$$\frac{}{\text{ITER } I \ / \{\} :: \ S \ \rightarrow \ S}$$

$$\frac{I\ /x\ ::\ S \rightarrow\ S'}{\text{ITER } I\ /\{x\ ;<\ tl >\ \} ::\ S\ \rightarrow\ \text{ITER } I\ /\{<\ tl >\ \} ::\ S'}$$

If a list l is concrete, the intructuion simply loops until the list ends with an empty list construction. Here we consider the synbolic execution of ITER when the list is a sybolic term or variable.

The sequence of intructuions I is a lamda function (lambda (ty : A) A) applied to the stack of type (ty : A), returning the stack of type A. We can represent I as the function f that takes an element of type ty and a stack of type A and returns a stack of type A.

$$f : \text{ty A}\ \rightarrow\ \text{A}$$

$$fx\ S = (\text{lambda (ty}\ \ : \text{A) A})(x\ ::\ S)$$

or

$$fx\ S = I/(x\ ::\ S)$$

Let $\{\ x\ ;\ <\ tl\ >\ \} :: S_0$ be the stack just before applying the instruction ITER I, where x be the head of the list and S_0 be the rest of the stack . For first loop, let S be f (x, S_0). S is the stack resulting from applying the function f to the element x and the stack S_0. In other words, the stack S is formed by applying the sequence of the instructions I to the stack $x :: S_0$.

Let the result of applying the instruction ITER I on the stack $l :: S$ be the stack Z, then Z is resulted from the *fold* function that applies the function f to each element of the list l and initial value is the stack S_0.

$$Z = \textit{fold}\ S_0\ fl$$

5 Constraints

$$\text{abs}(x) \geq 0$$

$$\text{size}(x) \geq 0$$

$$\text{len}(\text{slice}(byt, offset, len)) = len$$

$$\text{len}(x) \geq 0$$

6 Types

$$
\begin{aligned}
t ::= & \\
& \mid \langle\text{variable}\rangle \\
& \mid \langle\text{account constant}\rangle \\
& \mid \langle\text{int constant}\rangle \\
& \mid \langle\text{string constant}\rangle \\
& \mid \langle\text{byte sequence constant}\rangle \\
& \mid \text{Unit} \\
& \mid \text{True} \\
& \mid \text{False} \\
& \mid \text{Pair t1 t2} \\
& \mid \text{Left t} \\
& \mid \text{Right t} \\
& \mid \text{Some t} \\
& \mid \text{None} \\
& \mid \{\text{t ; ... }\} \\
& \mid \{\text{ Elt t1 t2 ; ... }\} \\
& \mid \{\langle\text{instruction}\rangle; \ldots\} \\
\langle\text{variable}\rangle ::= & \\
& \mid \mathbf{x} \\
\langle\text{account constant}\rangle ::= & \\
& \mid \text{balance} \\
& \mid \text{amount} \\
& \mid \text{sender} \\
& \mid \text{source} \\
& \mid \text{now} \\
& \mid \text{level} \\
& \mid \text{chain-id} \\
& \mid \text{self} \\
& \mid \text{self-address} \\
& \mid \text{total-voting-power} \\
& \mid \text{voting-power} \\
\langle\text{natural number constant}\rangle ::= & \\
& \mid \text{[0-9]+} \\
\langle\text{int constant}\rangle ::= & \\
& \mid \langle\text{natural number constant}\rangle \\
& \mid \text{-}\langle\text{natural number constant}\rangle \\
\langle\text{string constant}\rangle ::= & \\
& \mid \text{"}\langle\text{string content}\rangle\text{*"} \\
\langle\text{instruction}\rangle ::= & \\
& \mid \text{DROP} \\
& \mid \text{DROP}\langle\text{natural number constant}\rangle \\
& \ldots
\end{aligned}
$$

Fig. 2. Terms

$$
\begin{array}{rl}
p ::= & \\
 & \mid \langle \text{atomic formula} \rangle \\
 & \mid \neg\, p \;\mid\; p \,\wedge\, q \;\mid\; p \,\vee\, q \\
\langle \text{atomic formula} \rangle ::= & \\
 & \mid \langle \text{butop} \rangle \langle \text{bterm} \rangle \\
 & \mid \langle \text{bterm} \rangle \langle \text{biop} \rangle \langle \text{bterm} \rangle \\
\langle \text{butop} \rangle ::= & \\
 & \mid \text{not} \\
\langle \text{biop} \rangle ::= & \\
 & \mid \; = \;\mid\; > \;\mid\; < \;\mid\; >= \;\mid\; =< \;\mid\; != \\
 & \mid \text{and} \;\mid\; \text{or} \;\mid\; \text{xor} \\
\langle \text{bterm} \rangle ::= & \\
 & \mid t \\
 & \mid \langle \text{unop} \rangle t \\
 & \mid t\ \langle \text{binop} \rangle t \\
\langle \text{unop} \rangle ::= & \\
 & \mid \text{abs} \\
 & \mid \text{size} \\
 & \mid \text{int} \\
 & \mid \text{contract} \langle \text{type} \rangle \\
 & \mid \text{isleft} \\
 & \mid \text{isnone} \\
 & \mid \text{neg} \\
 & \mid \text{blake2b} \\
 & \mid \text{hash-key} \\
 & \mid \text{keccak} \\
 & \mid \text{pairing-check} \\
 & \mid \text{sha256} \\
 & \mid \text{sha3} \\
 & \mid \text{sha512} \\
 & \mid \text{implicit-account} \\
 & \mid \text{check-sig} \\
\langle \text{bitop} \rangle ::= & \\
 & \mid + \;\mid\; - \;\mid\; * \;\mid\; /
\end{array}
$$

Fig. 3. Predicates

$$\Gamma \vdash \mathbf{x} : \mathrm{ty} \qquad \Gamma \vdash \mathrm{Unit} : \mathrm{unit} \qquad \Gamma \vdash \mathrm{True} : \mathrm{bool} \qquad \Gamma \vdash \mathrm{False} : \mathrm{bool}$$

$$\Gamma \vdash \mathrm{balance} : \mathrm{mutez} \qquad \Gamma \vdash \mathrm{amount} : \mathrm{mutez} \qquad \Gamma \vdash \mathrm{sender} : \mathrm{address}$$

$$\Gamma \vdash \mathrm{source} : \mathrm{address} \qquad \Gamma \vdash \mathrm{now} : \mathrm{timestamp} \qquad \Gamma \vdash \mathrm{level} : \mathrm{nat}$$

$$\Gamma \vdash \text{chain-id} : \text{chain-id} \qquad \Gamma \vdash \mathrm{self} : \mathrm{contract\ ty}$$

$$\frac{\Gamma \vdash t_1 : \mathrm{ty}_1 \qquad \Gamma \vdash t_2 : \mathrm{ty}_2}{\Gamma \vdash \mathrm{Pair}\ t_1\ t_2 : \mathrm{pair}\ \mathrm{ty}_1\ \mathrm{ty}_2}$$

$$\frac{\Gamma \vdash t_1 : \mathrm{ty}_1}{\Gamma \vdash \mathrm{Left}\ t_1 : \mathrm{or}\ \mathrm{ty}_1\ \mathrm{ty}} \qquad \frac{\Gamma \vdash t_1 : \mathrm{ty}_1}{\Gamma \vdash \mathrm{Right}\ t_1 : \mathrm{or}\ \mathrm{ty}\ \mathrm{ty}_1} \qquad \frac{\Gamma \vdash t_1 : \mathrm{ty}_1}{\Gamma \vdash \mathrm{Some}\ t_1 : \mathrm{option}\ \mathrm{ty}_1}$$

Fig. 4. Typing rules for Terms

$$\Gamma \vdash \text{EXEC} : \text{ty}_1 \ :: \ \text{LAMBDA ty}_1 \text{ ty}_2 \ :: \ \text{A} \ \rightarrow \ \text{ty}_2 \ :: \ \text{A}$$

$$\Gamma \vdash \text{APPLY} : \text{ty}_1 \ :: \ \text{LAMBDA (Pair ty}_1 \text{ ty}_2\text{) ty}_3 \ :: \ \text{A} \ \rightarrow \ \text{LAMBDA ty}_1 \text{ ty}_2 \ :: \ \text{A}$$

$$\frac{\Gamma \vdash I_1 : \text{A} \ \rightarrow \ \text{B} \qquad \Gamma \vdash I_2 : \text{A} \ \rightarrow \ \text{B}}{\Gamma \vdash \text{IF } I_1 \ I_2 : \text{bool} \ :: \ \text{A} \ \rightarrow \ \text{B}}$$

$$\frac{\Gamma \vdash I_1 : \text{A} \ \rightarrow \ \text{B} \qquad \Gamma \vdash I_2 : \text{A} \ \rightarrow \ \text{B}}{\Gamma \vdash \text{IF-LEFT } I_1 \ I_2 : \text{or ty}_1 \text{ ty}_2 \ :: \ \text{A} \ \rightarrow \ \text{B}}$$

$$\frac{\Gamma \vdash I_1 : \text{A} \ \rightarrow \ \text{B} \qquad \Gamma \vdash I_2 : \text{B} \ \rightarrow \ \text{C}}{\Gamma \vdash I_1 \ ; I_2 : \text{A} \ \rightarrow \ \text{C}}$$

$$\frac{\Gamma \vdash I : \text{ty} \ :: \ \text{A} \ \rightarrow \ \text{A}}{\Gamma \vdash \text{ITER } I : \text{list ty} \ :: \ \text{A} \ \rightarrow \ \text{A}}$$

$$\frac{\Gamma \vdash I : \text{A} \ \rightarrow \ \text{bool} \ :: \ \text{A}}{\Gamma \vdash \text{LOOP } I : \text{bool} \ :: \ \text{A} \ \rightarrow \ \text{A}}$$

$$\frac{\Gamma \vdash I : \text{A} \ \rightarrow \ \text{B}}{\Gamma \vdash \text{DIP } I : \text{ty} \ :: \ \text{A} \ \rightarrow \ \text{ty} \ :: \ \text{B}}$$

$$\frac{\Gamma \vdash I : \text{A} \ \rightarrow \ \text{bool} \ :: \ \text{A}}{\Gamma \vdash \text{DIP } I : \text{bool} \ :: \ \text{A} \ \rightarrow \ \text{A}}$$

$$\Gamma \vdash \text{ADD} : \text{nat} \ :: \ \text{nat} \ :: \ \text{A} \ \rightarrow \ \text{nat} \ :: \ \text{A}$$

$$\frac{\text{len} A = n \qquad \Gamma \vdash I : \text{B} \ \rightarrow \ \text{C}}{\Gamma \vdash \text{DIP } n \ I : \text{A @ B} \ \rightarrow \ \text{A @ C}}$$

Fig. 5. Typing rules for Rules